AFRL-HE-WP-TR-2004-0113

United States Air Force Research Laboratory

Upgrading the ATB Model Code from FORTRAN 77 to FORTRAN 90

Thomas R. Gardner

BLACK ROCK DYNAMICS
30 Derussey Lane
Cornwell NY 12518

July 2004

Final Report for the Period January 1998 to May 2004

Human Effectiveness Directorate
Biosciences & Protection Division
Biomechanics Branch
2800 Q Street, Bldg 824, Rm 206
Wright-Patterson AFB OH 45433-7947

NOTICES

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise, as in any manner, licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

Please do not request copies of this report from the Air Force Resesarch Laboratory. Additional copies may be purchased from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Federal Government agencies and their contractors registered with Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Center
8725 John J. Kingman Rd., STE 0944
Ft Belvoir VA 22060-6218

TECHNICAL REVIEW AND APPROVAL

AFRL-HE-WP-TR-2004- 0113

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

FOR THE DIRECTOR

//Signed//

MARK M. HOFFMAN
Deputy Chief, Biosciences and Protection Division
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 2004	Final Report Jan 1998 – May 2004

4. TITLE AND SUBTITLE
Upgrading the ATB model code from FORTRAN 77 to FORTRAN 90

5. FUNDING NUMBERS
C-F33657-97-D-6004
PE: 62202F
PR: 7184
TA: 718402
WU: 71840202

6. AUTHOR(S)
Thomas R. Gardner

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BlackRock Dynamics
30 DeRussey Lane
Cornwall NY 12518

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory, Human Effectiveness Directorate
Biosciences and Protection Division
Biomechanics Branch
Air Force Materiel Command
Wright-Patterson AFB OH 45433-7947

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-HE-WP-TR-2004- 0113

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
The Articulated Total Body (ATB) Model and its predecessors have been in use for over twenty-five years to study the effects of various force environments on the response of a multi-body subject. The ATB model has been used by both government and industry to study the effects of automobile crashes, aircraft ejections, and other accident scenarios. The applications of the ATB Model have varied since its inception, and it is now commonly used for accident reconstruction, ergonomic simulations and a myriad of other situations where human body motion is modeled. The ATB Model was originally written in the FORTRAN IV (FORTRAN 66) programming language, but with time the code has gradually been modified to become essentially Fortran 77 compliant code. This modernization of the code has helped to reduce some of the clever coding logic originally employed to circumvent some of the hardware constraints of the earlier computers, such as purposely allowing arrays to overflow in adjacent memory locations to conserve memory space. With the increased use of the ATB Model, it would be beneficial to have the program coded in a more modern modular format that is readily amenable to future enhancements. Such a modular version of the code would also easily interface with other programs, such FEM and aerodynamic codes, other dynamics codes, feedback control packages, graphics packages, and numerous other related codes. In addition, such a modular code would lend itself to faster debugging and reduce the potential for undetected errors. The Fortran 90 programming language was chosen as the upgrade path for the ATB Model code for a number of reasons. Since Fortran 90 is a superset of Fortran 77, any standard Fortran 77 code should run without modification. In addition, anyone familiar with Fortran 77 can quickly become knowledgeable about Fortran 90. A Fortran 77-compliant program can be first compiled using the Fortran 90 compiler and can then be gradually and incrementally upgraded without having to completely rewrite the code.

14. SUBJECT TERMS
FORTRAN 90, ATB, Human Body Modeling, Simulation

15. NUMBER OF PAGES
121

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	10. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UNLIMITED

THIS PAGE INTENTIONALLY LEFT BLANK.

PREFACE

The research in this report was conducted by Thomas R. Gardner of BlackRock Dynamics under a subcontract to General Dynamics (FY98-06-08) for the Biomechanics Branch, Biosciences and Protection Division, Human Effectiveness Directorate of the Air Force Research Laboratory.

Dr. Joseph A. Pellettiere of AFRL/HEPA was the technical monitor for the contract while Ms. Annette Rizer and Mr. Huaining Cheng of General Dynamics provided invaluable editing, review, and coordination of the programming tasks.

THIS PAGE INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

This report is a summary of the work performed for General Dynamics under Consulting Agreement FY98-06-08. A brief overview of the need for this work and the approach taken is given below.

1.1 Background

The Articulated Total Body (ATB) Model and its predecessors have been in use for over twenty-five years to study the effects of various force environments on the response of a multi-body subject. The ATB Model is a U.S. Air Force sponsored, much-enhanced derivative of the Crash Victim Simulator (CVS) model developed by Calspan for the DOT and MVA [1]. The CVS model was originally developed to model crash dummies and has been used by both government and industry to study the effects of automobile crashes [2], aircraft ejections [3], and other related transportation accident scenarios [4]. The applications of the ATB Model have become more varied since its inception, and it is now commonly used for accident reconstruction, ergonomic simulations and a myriad of other situations where human body motion is modeled.

1.2 Justification

The ATB Model was originally written in the FORTRAN IV (FORTRAN 66) programming language, but with time the code has gradually been modified to become essentially Fortran 77 compliant code. This modernization of the code has helped to reduce some of the clever, yet obtuse and convoluted coding logic originally employed to circumvent some of the hardware constraints of the earlier computers, such as purposely allowing arrays to overflow in adjacent memory locations to conserve memory space. However, the ATB Model code still remained, in many sections, dense, hard to follow, and very difficult to debug. Furthermore, this older style of programming made it very difficult to modify the code to any extent and made it cumbersome, even for people familiar with the code, to make significant additions and modifications to the code. With the increased use of the ATB Model, it would be beneficial to have the program coded in a more modern modular format that is readily amenable to future enhancements. Such a modular version of the code would also easily interface with other programs, such FEM and aerodynamic codes, other dynamics codes (e.g. the Head-Spine Model), feedback control packages, graphics packages, and numerous other related codes. In addition, such a modular code would lend itself to faster debugging and reduce the potential for undetected errors.

The standard input file (denoted by the extension .ain) to the ATB Model is a text file with each record in the file corresponding to an 80 column Fortran data card. Therefore, each record or consecutive sequence of records corresponds to a specific Fortran formatted sequential READ statement. This arrangement requires strict placement of input data into the correct columns.

Placing a decimal point or digit into a wrong column may crash the simulation, or even worse, produce erroneous results that are hard to detect and debug. In addition, the maximum 80 input column limit imposed by Fortran 77 greatly restricts the input space for some variables. Many of them are allocated with only 6 digits. When encountered with a very small negative input value, the user may only be able to keep at most two significant digits because the negative sign and Fortan scientific E editor will take up at least 4 digits. To eliminate these drawbacks, a new free format input structure is needed, as well as the capability to convert existing .ain files to a free format input structure.

1.3 Rationale

The Fortran 90 programming language was chosen as the upgrade path for the ATB Model code for a number of reasons. Since Fortran 90 [5,6] is a superset of Fortran 77, any standard Fortran 77 code should run without modification. In addition, anyone familiar with Fortran 77 can quickly become knowledgeable about Fortran 90. A Fortran 77-compliant program can be first compiled using the Fortran 90 compiler and can then be gradually and incrementally upgraded without having to completely rewrite the code. This approach greatly reduces the potential for inadvertent coding mistakes that inevitably arise when any code as large as the ATB Model is completely rewritten. Furthermore, there are a number of Fortran 90 compliant compilers available both for personal computers as well as for workstations and supercomputers. The personal computer compilers are very easy to use and reasonably priced.

Fortran 90 is an object-oriented language that allows for dynamic memory allocation. It produces fast, efficient machine code and has provisions that allow the programmer to control the numerical precision of the code across platforms. Fortran 90 has features that facilitate easier linking with C++ subroutines and Fortran 95 [7,8] has intrinsic features that permit parallelization of the code. Several additional features available in Fortran 90 are modules, structures, recursion, pointers, allocatable arrays, definable operators, CASE, CYCLE and EXIT constructs and variable names that may be up to 31 characters in length. Many of these new features were utilized in the upgrade of the ATB Model code.

1.4 Approach

This effort has attempted to address the concerns of code readability, portability and adherence to modern programming practices by upgrading the ATB Model to be a Fortran 90/95 compliant code, henceforth know as Version V.3 of the ATB Model. All subroutines within the code were modified to some extent, some much more than others. New subroutines were written and many long subroutines were broken into smaller, more functionally self-contained subroutines. Many modifications were made to the ATB Model code, including correcting some

previously undetected bugs. A new type of free format input file was added utilitizing Fortran 90/95 list-directed input, and an option was built into the program to convert input from the old fixed format into the new free format. Overall, the changes made to the code were primarily programming oriented. Every effort was made to not alter the underlying logic used for the mathematical/physical algorithms for models, such as the plane/segment force interaction, harness-belt model, etc. This report details these changes.

2. CODING CHANGES TO VERSION V.1

Numerous changes were made to the ATB Model Version V.1 code. Some changes were global, affecting all subroutines within the code, whereas other changes were specific to individual subroutines.

2.1 Task Dictated Global Coding Changes

The global changes explained in the sections below pertain to the global changes as required in the Statement of Work

2.1.1 Elimination of Common Blocks

The most significant global change was to remove all common blocks and replace all the variables that were contained within the common blocks with global variables that are contained in modules. The names of most of the variables that had been in the common blocks remain the same, with only a few variables given modified names to eliminate duplicate names with local variables. Common blocks were eliminated to permit the use of the USE ONLY option of the USE statement. If a common block is placed in a module, and that module is accessed by a subroutine via the USE statement, all the variables contained within the named common block become accessible to the subroutine. It is not possible to limit the scope of the subroutine to only those variables used within the subroutine that are also contained within the named common block.

The USE ONLY option in Fortran 90 allows the subroutine to have access to only those variables listed in the USE ONLY statement. By listing in the USE ONLY statement only those variables needed by a subroutine, it becomes easier to understand where global variables are altered/modified throughout the code. It also makes it easier to understand which global variables are needed by a subroutine. Therefore, to take advantage of the USE ONLY option in Fortran 90, the use of common blocks was eliminated.

All common blocks that were in Version V.1 were placed in one of three modules according to function. Each of the modules is described below, with a list of the named common blocks they contain. Each module also contains comment statements that contain the format of the common block as it existed in the Version V.1 code to make it easier for ATB Model programmers to follow the coding changes. The variables that had been contained within the named common blocks are grouped together where they are typed to further aid in understanding the changes that were made to the code.

2.1.1.a MODULE_STANDARD

This MODULE contains all of the common blocks that existed in the versions of the code prior to Version V.1. It also contains several common blocks added in Version V.1 for the robotics option and for additional time history output for the total body properties (Card H.10). It does not contain the common blocks related to the deformable segment option or to the water forces. Variables from the following named common blocks are included in MODULE_STANDARD

ABDATA	CNSNTS	CSTRNT	HRNESS	TMPVS2
ACTFR	CNTSRF	CYDATA	INTEST	VPOSTN
ACTFR1	COMAIN	DAMPER	JBARTZ	WINDFR
BAGDIM	CONTRL	DESCRP	RSAVE	XTRA
CDH10C	COUT	FILEN	SGMNTS	
CDINT	COUTFMT	FLXBLE	TABLES	
CEULER	COUTN	FORCES	TEMPVI	
CMATRX		HBPTRB	TITLES	

2.1.1.b MODULE_FLEXIBLE

This MODULE contains all of the common blocks that pertain to the deformable segment option. Variables from the following named common blocks are included in MODULE_FLEXIBLE.

FXBODY	FXINT	FXOUT	FXXTRA
FXCOEF	FXJROT	FXSING	OLDDAT
FXFRC	FXNVEL	FXVAR	

2.1.1.c MODULE_WATER

This MODULE contains all of the common blocks that pertain to the water forces option. Variables from the following named common blocks are included in MODULE_WATER.

ELPDAT	WATINF2	WMASS
WATGRD	WAVEDAT	WRESLTS
WATINF1	WFACOP	TEMPFD

2.1.1.d Advantages of MODULES

The use of modules to store global variables has several advantages over the BLOCK DATA technique used in previous versions of the model. First and foremost is that a global variable is typed and dimensioned only once, in the module, and the typing and dimensioning remain the same throughout the program. This consistency is enforced during the compilation of the program. Previous uses of INCLUDE statements (a common, but nonstandard extension of Fortran 77) did not provide this consistency. Variables from a common block included in a subroutine could be used differently, i.e. as a different type within a subroutine because no mechanism exists in Fortran 77 to check variable usage across subroutines.

Parameters specified in a module are available to all subprograms via the USE statement. This allows for easy, consistent specification of dimensions throughout the program that is enforced during compilation.

The KIND of the variable may also be specified in the module (as will be explained in more detail below) as a parameter. Changing of a single parameter in the module and recompiling the code allows entire sets of variables to be changed from single precision to double precision, or visa versa, with no additional coding changes.

2.1.1.e Elimination of BLOCKDATA

The use of modules has negated the need for the BLOCKDATA subprogram that existed in previous versions of the model. Three modules replace the function of the BLOCKDATA subprogram, but the use of modules has significant ramifications beyond the advantages mentioned above. Previous versions of the code, in an attempt to reduce memory space, purposely allowed some common blocks to overflow into adjacent memory. This scheme worked because the data in the overwritten areas of memory were no longer needed by the program at the point in the execution of the program when they were overwritten. This technique had worked because the common blocks were in a specific and necessary order in the BLOCKDATA subprogram. The effect of the BLOCKDATA subprogram was to set aside a block of virtually contiguous memory large enough to contain all the elements in the BLOCKDATA subprogram. However, this was based on the assumption of static memory allocation, i.e., this block of memory was set aside when the program was initially loaded into memory and remained the same until the program completed.

This logic prevented the use of the more modern and efficient technique of dynamic memory allocation, thereby requiring the use of the '/static' option on most compilers. Furthermore, it required that programmers understand the flow of the program if the ordering and dimensioning of the common blocks were to be changed. In addition, the use of the helpful 'array bounds checking' feature in most debuggers could not be used because the dimensions of many arrays were purposely being exceeded. Additional global variables (to be detailed below) and the use of allocatable variables (to be explained in more detail below) have eliminated the need for the overwriting of any arrays. With individual global variables in the module replacing common blocks, the need for the ordering of the common blocks and hence the variables within them is no longer relevant. The computer operation system can make optimal use of dynamic memory allocation (see caveat below regarding the referencing of array elements), programmers need no longer to be concerned about the ordering of existing or new variables in the code, and the use of the 'array bounds checking' feature in many debuggers can now be used.

2.1.2 Dimensions of Arrays Set by Parameters

The dimensions of all global arrays are set in three modules and are enforced consistently across all subroutines that reference the modules during compilation. Many of the dimensions are set by parameters that are also typed and defined in one of the modules, usually in MODULE_STANDARD. There are, however, a number of arrays that are still explicitly dimensioned. To improve the ease of changing code size limitations, these arrays should eventually be dimensioned with parameters.

2.1.3 Elimination of COMMON /TEMPVS/

The use of the named common block /TEMPVS/ was eliminated from all subroutines throughout the program. This named common block served several functions in Version V.1 of the code. In some subroutines it was used to provide scratch space for local variables used only in that subroutine. In other places it was used to act as a named common block that stored information that was shared by several subroutines. This shared function use was used in several places in the code but by different groups of subroutines.

The original intent of using /TEMPVS/ was to reduce the memory requirements of the early versions of the ATB Model in the days when core memory was very limited. The trade-off was that it made understanding and modifying the code very difficult. With the advent of newer machines with much larger memories, the cost of using /TEMPVS/ came to vastly outweigh its benefits. Because variables varied in both size and type between different subroutines that used /TEMPVS/, memory alignment errors and problems occurred on many machines. Dynamic memory allocation was not possible and the overall coding scheme used for /TEMPVS/ was neither Fortran 77 nor Fortran 90/95 compliant. Numerous problems resulted from previous coding modifications that did not properly account for how and where data were shared between the subroutines using /TEMPVS/. Furthermore, it reduced the modularity of the code and made it much more involved to modify any of the subroutines that used /TEMPVS/.

The elimination of /TEMPVS/ from a subroutine or group of subroutines was handled in several different ways, depending on what function or functions /TEMPVS/ served in that subroutine. Where /TEMPVS/ was used to act as scratch space for local variables for a subroutine, these variables were made into local variables, and in most cases given the same name as before, or given the same name with the suffix "_LCL" added if the name conflicted with an existing global variable of the same name.

When /TEMPVS/ was used to share data between several subroutines, the elimination of /TEMPVS/ was much more difficult and was effectively accomplished in two stages. First, a new named common block was created for each group of subroutines that shared data. Once the

program was debugged, all of the newly formed common blocks were eliminated and the variables they contained were made into global variables defined in MODULE_STANDARD. Usually the original name of the variable was kept as the global variable. However, when there was a conflict with a previously existing global variable with the same name, the variable had a suffix appended to it. As an example, the newly created global variables from the /TEMPVS/ variables associated with the harness belt subroutines had the suffice "_HRN" appended to their original name if their names conflicted with existing global variables. In several cases, different subroutines sharing variables had referred to them by different names. To help previous users of the code track the changes made to the code, the rename option of the USE statement was used rather than the cleaner approach of renaming the variables within the involved subroutines. In a few instances where data were passed between subroutines using /TEMPVS/, and if the data were not large arrays, the variables were passed as arguments between the two subroutines.

Though this approach increased the memory requirements for the program, the benefits gained far outweigh the increased memory requirements even if static memory allocation is assumed. The new form of the code permits the use of dynamic memory allocation, so the overall increase in memory requirements many not be substantial, and depending on how memory is handled, might actually decrease.

Many comments are provided in MODULE_STANDARD detailing these changes, with the global variables marked off in sections by the intermediate named common block names. These intermediate names are provided below, with the names of the subroutines from Version V.1 that had shared data via /TEMPVS/. It is repeated that these common blocks no longer exist and were used only to help in the transition of the code to its present form. Their names are provided below only to help former users of the code to understand how the code was modified. A listing of all the global variables that were associated with each of the temporary, intermediary, named common blocks can be found in the comments in MODULE_STANDARD (Appendix A.1).

Temporary Intermediary Named COMMON	Version V.1 Subroutines Sharing Data
/AIRBAG_TEMPVS/	AIRBAG, AIRBGG
/BELT_TEMPVS/	BELTG, BELTRT
/CINPUT_TEMPVS/	CINPUT, FDINIT, FINPUT, HINPUT, KINPUT
/DAUX_TEMPVS/	DAUX11, DAUX12, DAUX22, DAUX31, DAUX32, DAUX33
/HEDING_TEMPVS/	HEDING, POSTPR
/HRN_TEMPVS/	HPTURB, HBPLAY, HBELT, HSETC

/PLELP_TEMPVS/	HYEST, HYLPX, HYNTR, PLEDG, PLELP, PLSEGF, SEGSEG
/VIN_TEMPVS/	VINPUT, VINO12, VINO34, VSPLIN, VINTST

2.1.4 Elimination of Implicit Typing of Variables

The IMPLICIT NONE statement was included in all subprograms throughout the program. This forced all variables to be explicitly typed, rather than using default typing or the IMPLICIT typing statements in previous versions of the code. Implicit typing of variables is generally considered to be a poor programming technique that can result in multiple programming errors. As explained in Section 2.1, all global variables were explicitly typed in the modules, with the compiler enforcing this typing in any subroutine that used a global variable from the module. All local variables in each subroutine were also explicitly typed. The advantages of local explicit typing are twofold. First, it reduces programming errors by flagging any variables whose names were mistyped rather than setting the value of the variable to 0. Second, it makes it clear to the programmer what local variables are used within the subroutine by providing a list of all local variables used in the subroutine at the beginning of each subroutine.

2.1.5 Elimination of Hollerith Characters

All Hollerith edit descriptors were replaced by character string edit descriptors, delimited by single quotes, in FORMAT statements throughout the code. Hollerith edit descriptors are an outdated, error-prone edit descriptor type that is not supported in Fortran 95.

2.1.6 Replacement of REAL Variables with CHARACTER Variables

Because the ATB Model code was developed before the CHARACTER type variable was available in Fortran, the program used single and double precision variables to handle character information. The use of single and double precision real variables to represent character information resulted in awkward, difficult to follow and error-prone code. All instances of this have been eliminated, with only CHARACTER type array variables used to handle character information. The following global variables were changed to type CHARACTER. Note that the /TEMPVS/ common block variables differ by subroutine. Variables denoted by a "+" pertained to subroutine HEDING in Version V.1. Variables denoted by a "*" pertained to subroutines CINPUT, FDINIT, FINPUT, HINPUT and KINPUT in Version V.1. Several other minor character related changes were associated with the elimination of the /TEMPVS/ common blocks but are not explained here. In addition, several subroutines have character related changes that are specific to that subroutine and are not explained here.

COMMON BLOCK	Variable(dimension)	Old Type	New Type(dimension)
/CNSNTS/	UNITL	REAL*8	CHARACTER*8
" "	UNITM	REAL*8	CHARACTER*8
" "	UNITT	REAL*8	CHARACTER*8
/INTEST/	REGT(4*MAXSEG)	REAL*8	CHARACTER*8(4*MAXSEG)
" "	SEGT(4*MAXSEG)	REAL*4	CHARACTER*4(4*MAXSEG)
/TEMPVS/+	AHED(5,2)	REAL*4	CHARACTER*20(2)
" + "	AHEAD(5,20)	REAL*4	CHARACTER*20(20)
" + "	BLANK	REAL*4	CHARACTER*4
" + "	GHED(2)	REAL*4	CHARACTER*4(2)
" + "	HEADJJ(4,2)	REAL*8	CHARACTER*32(2)
" + "	HEADR(20)	REAL*8	CHARACTER*8(20)
" + "	HEDJ(4,2)	REAL*8	CHARACTER*32(2)
" * "	JTITLE(5,51)	REAL*4	CHARACTER*20(51)
" * "	KTITLE(31)	REAL*4	CHARACTER*4(31)
" + "	PHED(5)	REAL*4	CHARACTER*4(5)
" + "	RHED(3)	REAL*8	CHARACTER*8(3)
/TITLES/	DATE(3)	REAL*4	CHARACTER*12
" "	COMENT(40)	REAL*4	CHARACTER*160
" "	VPSTTL(20)	REAL*4	CHARACTER*80
" "	BDYTTL(5)	REAL*4	CHARACTER*20
" "	BLTTTL(5,8)	REAL*4	CHARACTER*20(8)
" "	PLTTL(5,MAXPLN)	REAL*4	CHARACTER*20(MAXPLN)
" "	BAGTTL(5,6)	REAL*4	CHARACTER*20(6)
" "	SEG(MAXSEG)	REAL*4	CHARACTER*4(MAXSEG)
" "	JOINT(MAXJNT)	REAL*4	CHARACTER*4((MAXJNT)

2.1.7 Use of Longer Variable Names

The use of longer variable names to aid in the understanding of the code was done only very sparingly for two reasons. First, to make it less confusing for readers of the code trying to understand the differences between Version V.1 and Version V.3. Second, it is hoped that future versions of the code will make use of structures, making any renaming of the variables for this effort short-lived because the use of structures would necessitate a full-scale renaming effort throughout the entire code. Longer variable names were used primarily for new variables that were introduced to the code in this effort. The other significant use of longer variable names was to append the suffix "_LCL" to local variable names in subroutines where the local variable originally had the same name as a global variable but where the global variable was not scoped (referenced) by the subroutine. Since these subroutines did not scope the global variable, there was no conflict between the local variable and global variable of the same name. However, the use of the same name for a global and local variable is a poor programming technique and has a high potential for error.

2.1.8 Use of the INTENT Statement

All new subroutines made use of the INTENT statement, i.e. INTENT(IN), INTENT(OUT), or INTENT(INOUT). The INTENT statement explicitly specifies how an argument that is passed

to the subroutine or function is to be used by the subprogram. INTENT(IN) means that the argument may only be used, it may not be altered. INTENT(OUT) means that the argument is defined within the subprogram. No value is available for it upon entrance into the subprogram. INTENT(INOUT) means that the argument may have either or both characteristics. The advantage of the specifying intent is that it makes the usage and purpose of the passed arguments clearer to readers of the code, hence aiding in understanding of the code. Furthermore, the intent of each argument is checked during compilation and any inappropriate use of the argument is flagged as an error. This forces the arguments to be used as intended within that subprogram.

2.1.9 Elimination of Arithmetic IF Statements

All arithmetic IF statements throughout the code were replaced with IF-THEN constructs. The use of the arithmetic IF statement makes it hard to follow the logic of a subroutine and can lead to coding errors. Furthermore, the use of the arithmetic IF statement is not supported by standard Fortran 95.

2.1.10 Elimination of Nested DO's Terminating on the Same Statement

In keeping with modern coding practices, and Fortran 90 guidelines, all nested DO loops throughout the program that terminated on the same statement were replaced with a DO construct that had each DO loop terminating on a unique statement. This practice makes the code easier to follow and more modular in nature.

2.1.11. Proper DO Termination Statements

All DO loops throughout the program that did not terminate on either a CONTINUE statement or an END DO statement were modified to do so. This again is in keeping with modern coding practices, reduces the potential for logic errors, and makes the code more modular.

2.1.12. Use of END DO Statements

Many numbered DO loops throughout the program were replaced with DO-END DO constructs, with proper indentation, to make the code easier to follow and more modular in nature. Long DO loops (effectively DO loops whose code would span more than about a full screen of source listing) were left as numbered DO loops to make it easier to follow the logic of the DO loop across one or more pages (screens) of source code listings.

2.1.13 Arrays Initialized as Objects

Scattered throughout Version V.1 of the code are instances where multiple, nested DO loops are used to initialize multi-dimensional arrays. In selected locations where the entire array was being initialized, the object oriented nature of Fortran 90 was utilized, e.g. the multi-dimensioned global array **A** was initialized to 0 by the simple statement **A = 0.0**. This avoids the need for

knowing the number of dimensions and the span of the array within the subroutine. The dimension and size of the global array can be modified in the module, yet there is no need to modify DO loop indices within the subroutine. The usefulness of this feature is diminished if only parts of the array are being defined and consequently this feature was not applied in these cases.

2.1.14 Use of Allocatable Arrays

Allocatable arrays are arrays in Fortran 90/95 that are designated in the beginning of the subroutine as being able to utilize dynamic memory allocation. When the array is used in the subroutine, memory is allocated for it. When the array is no longer needed, memory can then be deallocated, making the memory again available to the system. In general, the present structure of the ATB Model prevents the efficient use of this feature. There were, however, two locations within the code where this feature was used.

In Version V.1 of the ATB Model the large named common block, /TEMPVS/, was shared by the subroutines associated with vehicle motion. Several arrays used /TEMPVS/ to store the large amounts of vehicle input data needed for the six-degree-of-freedom spline-fit option (option 4). Since these data are needed only to compute the vehicle motion at the beginning of the simulation, the relevant arrays are made allocatable in Subroutine VSPLIN. The arrays are then allocated when the subroutine is entered, used only in that subroutine and then deallocated once the vehicle motion is computed. Upon deallocation, the memory utilized by these large arrays is then made available to the system for other uses.

An allocatable array is also used in Subroutine POSTPR to store tabular time history data used in the computation of injury criteria. These data are then passed to Subroutine HICCSI to perform the actual calculation of the injury criteria. Though the amount of memory saved by the use of an allocatable array in Subroutine POSTPR is not that substantial, an allocatable array was used in here to serve as another example of how allocatable arrays could be used in the ATB Model code.

2.1.15 Variable Logical Unit Numbers

All references to explicit logical unit numbers were replaced with integer variable names. This feature allows easy redirection of program input and output if logical unit number conflicts arise when the ATB Model is linked with other programs, such as graphics or finite element programs. The logical unit numbers are defined in MODULE_STANDARD and are listed below.

Variable Name	Function	Current Value
LULIN	list-directed standard input unit	3
LUAIN	fixed-format standard input unit	5
LUAOU	standard output unit	6
LUTP8	unformatted file for postprocessing	8
LUVIEW	graphics output unit	1

LUFLEX	flexible segment input	11
LUTERM_IN	terminal/console input	5
LUTERM_OUT	terminal/console output	0

2.1.16 Use of Structures

The coding associated with the control of the printing of individual tabular time history output by NPRT(18) on Card A.5 was completely revised. A structure was created for each type of tabular time history in MODULE_STANDARD using the user defined derived type TTH_DESCRIP which contained three components, BEGIN_LUNUM, END_LUNUM, PRINT. BEGIN_LUNUM is the beginning logical unit for the tabular time history, END_LUNUM is the last logical unit number for the tabular time history and PRINT is a logical variable controlling whether the tabular time history is to be printed. The value of PRINT is deduced from the value of NPRT(18) and set in Subroutines LUNUM_HCARDS and LUNUM_FORCES. The use of this structure makes the code more modular and easier to follow. An additional structure of derived type OUTPUT_TIMES was created for use with the outputting of simulation data. It is defined in MODULE_STANDARD and its associated components can found listed in the code in Appendix A.1.

A structure, of derived type SEGMENT, was created to encompass all the parameters associated with the segments. It is defined in MODULE_STANDARD and its associated components can found listed in the code in Appendix A.1. Comments contained within the code show the association between the variable names previously used in the code and the corresponding component names in the structure.

A structure, of derived type JOINT, was created to encompass all the parameters associated with the various types of joints available in the model. It is defined in MODULE_STANDARD and its associated components are listed in the code in Appendix A.1. Comments contained within the code show the association between the variable names previously used in the code and the corresponding component names in the structure. Not all joint parameters have been incorporated into the JOINT structure. Suggested names for these parameters have been included as comments in the source code, but have not yet been implemented.

2.1.17 Additional Comments

Additional comments were added throughout the code to improve the readability of the code. A limited number were added to existing subroutines, with ample comments included in all new subroutines. To further improve code readability, all comments throughout the code were changed from upper case to lower case with, in most cases, grammatically correct sentence structure and capitalization. All program variable names were fully capitalized to further aid in understanding of the code.

2.2 Additional Global Coding Changes

Several global coding changes were made in addition to the task dictated global changes. These changes provide additional features and enhancements to the ATB Model.

2.2.1 Renamed Local Variables Conflicting with Global Variables

All local variables having the same name as a global variable (variables contained in MODULE_STANDARD, MODULE_FLEXIBLE, or MODULE_WATER) were renamed, even if the subroutine containing the local variable did not scope any of the modules. In general, though not always, the new variable name was created from the original variable name with the suffix "_LCL" appended. This was done to maintain ease in understanding the code for programmers all ready somewhat familiar with the code. The variables were renamed to avoid scoping problems that could arise should the subroutine in question scope one of the modules with the USE or USE ONLY statements. If the name had not been modified and the USE statement included in the subroutine, what originally was a local variable would now scope to the global variable, most likely unnoticed by the programmer. Hence any modifications to the local variable would now be felt globally, possibly resulting in unforeseen consequences throughout the code. If the programmer mistakenly assumed that the local variable was referring to the global variable of the same name and purposely included the local variable in a USE ONLY statement, the same problems could occur.

2.2.2 Elimination of GO TO Statements

All GO TO statements were eliminated, the most time-consuming and labor intensive of any individual global program modification. The GO TO statements were replaced with modern constructs, usually with IF-THEN blocks. The justifications for these changes were many-fold. The GO TO construct has a very high potential for coding and/or logic mistakes during future modifications to the code. This type of construct is very difficult to follow; it makes the code very hard to read and is in general strongly discouraged by modern program practices. The IF-THEN block construction, without specific label numbers and proper indentation makes the code more modular, easier to read and much easier to debug. It also facilitates future coding changes.

However, the replacement of the GO TO construct with more modern constructs, such as the IF-THEN block, might slightly alter the underlying assembly language after compilation, as was found during previous modifications to the code[9], especially when different levels of optimization are used. This slightly different flow of the program at the machine level could result in slightly different results because of numerical round-off and finite machine precision. Very small differences were observed, as explained in Section 3.4, which may be partially attributable to these coding changes.

2.2.3 Segregated BSF Array

The global array BSF, which in Version V.1 had been stored in common block /FORCES/, was used to store data for the tabular time history outputs. It contained data for the simple belts, the harnesses, and the spring dampers. The data were stored sequentially in the array. The counter variable NBSF was used to keep track of data storage across subroutines. However, the option to individually limit the force-related tabular time history data (Card A.5, NPRT(18)) had an error that would not print the spring damper data out properly if both harness belts and spring dampers were used in a simulation but only the spring damper output was desired.

To correct this error and improve the modularity of the code, the array BSF was replaced in the module MODULE_STANDARD and throughout the code with three individual arrays, HARNESS_FORCE for the harness data, BELT_FORCE for the simple belt data and DAMP_FORCE for the spring damper data. In addition, parameters were used to dimension arrays for the number of spring dampers and simple belts permitted in a simulation. The computed data are always stored in the respective arrays if spring dampers, harnesses or simple belts are specified in a simulation, even if the data output is not desired. It is felt that this approach is more consistent with the general logic of the program and provides more flexibility for future code modifications.

2.2.4 Use of KIND Specification Throughout Code

Fortran 90 provides a new feature, the specification of the KIND of a variable. Each intrinsic type of variable can be given more than one type of representation, such as single and double for type REAL and INTEGER*2 and INTEGER*4 for type INTEGER. The representation of a variable (the number of bits used internally to represent a certain variable type) determines the level of precision of the stored data. Therefore the KIND of an intrinsic type affects the precision of the data being manipulated by the code. Fortran 90 also provides a number of intrinsic functions that allow the programmer to assess and control the numeric precision of a machine. The intent of these new features and additional intrinsic functions is to allow the programmer to include in the program code that will control the numerical precision of the calculations across platforms and operating systems. This ensures that simulations run with identical input will produce essentially identical output for different operating environments, an extremely useful and necessary feature for numerically intensive codes such as the ATB Model.

As a big first step in allowing for the control of the ATB Model's numerical computations, all variables through the code were explicitly given a KIND. A PARAMETER was specified for each of the types used through the code, as shown below.

ICHAR_STD = 1	1 character of this type is contained in 1 byte
INTEGER_STD = 4	1 integer of this type is contained in 4 bytes

IREAL_HIGH	= 8	1 real of this type is contained in 8 bytes
IREAL_STD	= 4	1 real of this type is contained in 4 bytes
LOGICAL_STD	= 4	1 logical of this type is contained in 4 bytes

The use of LOGICAL*1 was eliminated throughout the code as a nonstandard extension of Fortran 90/95 and replaced with LOGICAL_STD (LOGICAL*4). The majority of the code currently uses double precision computations. However, if one desired to change the use of double precision to single precision, only IREAL_HIGH need be changed from 8 to 4 and the code recompiled. Probably of more interest to most users however, is the use of the quad precision option on some machines (this option is not currently implemented in the Fortran compiler for Intel machines). All that is needed to use the quad precision is to change IREAL_HIGH from 8 to 16 on such machines, recompile the code and computations and data handling would then be properly done in quad precision.

2.2.5 Use of Generic Intrinsic Subroutines

Fortran 90 provides intrinsic subroutines that are generic in nature. These subroutines are effectively interface subroutines that will call the proper underlying subroutine based on the type of the argument passed to the generic subroutine. An example is the generic routine to compute the sine function. The generic call is SIN(X). If X is real it will call the actual SIN(X) subroutine for real values. If X is double precision, the actual subroutine DSIN(X) will be called and if X is complex, the actual subroutine CSIN(X) will be called.

The advantage of generic intrinsic subroutines is that one can change the KIND of the variables, as explained in Section 2.2.4, without needing to go through the code and change the name of each intrinsic subroutine. Since the use of the KIND option was implemented throughout the code, all calls to intrinsic subroutines throughout the code were changed to generic intrinsic calls. Listed below are all the intrinsic subroutine names in Version V.1 and the generic name that was used to replace it.

V.1	F90	V.1	F90
DABS	ABS	DSIN	SIN
DACOS	ACOS	DSQRT	SQRT
DASIN	ASIN	DTAN2	ATAN2
DCOS	COS	DTANH	TANH
DEXP	EXP	IABS	ABS
DLOG	LOG	IDINT	INT
DMIN1	MIN	MAX0	MAX
DSIGN	SIGN	MIN0	MIN

In addition, the intrinsic function FLOAT is not supported by Fortran 90 and was replaced by the intrinsic function REAL. The intrinsic function DBLE was also replaced by the intrinsic function REAL to make use of the KIND specification permitted in the function REAL but not in the function DBLE.

2.2.6 Listing of Files Used By Simulation

The filenames and full path of all the input and output files used by a simulation are now outputted to the .AOU file. In addition, the type of the tabular time history contained in a file is also listed by providing the range, i.e. *filename.t*xxx through *filename.t*yyy, where xxx is the starting number of that type of tabular time history and yyy is the ending number of that type of tabular time history. This additional data provide a record of run results and aids in the use of the multiple tabular time history option. New subroutines LUNUM_FORCES, LUNUM_HCARDS and OUTPUT_LUNUM were added to provide this feature.

2.2.7 Time and Date Stamping

The feature to time and date stamp the start and end of the run was provided using the Fortran 90 standard intrinsic subroutine DATE_AND_TIME. Hence this option will work on any Fortran 90 compliant machine. The new subroutines CONVERT_TIME, DATE_TIME and GET_MONTH_NAME were added to support this feature.

2.2.8 Subroutine Specific CPU Calculations

Subroutines ELTIME and L_LTIME were modified to work correctly, i.e. provide a tally of the CPU time used by selected subroutines. (In older versions of the code, L_LTIME had been named LTIME but it was renamed to avoid conflicts with Unix intrinsic functions on some machines that had the same name.) Two versions of this change were developed, one which works on Fortran 95 compliant machines and calls the Fortran 95 intrinsic subroutine CPU_TIME and one which provides slightly less information and works on all Fortran 90 compliant machines using the Fortran 90 intrinsic subroutine DATE_AND_TIME. The user selects which version of the subroutine is linked with the code depending on the compiler used. The COMPAQ Visual Fortran for the PC is Fortran 95 compliant. The MIPSpro F90 compiler for the Silicon Graphics R10000 chip machines is F90 compliant.

2.2.9 Long Subroutines Broken Up by Function

Many of the excessively long subroutines were broken up into several smaller subroutines. The smaller subroutines were more modular in nature, with each usually performing a specific task. This is in keeping with modern coding practices, makes the code much easier to understand and follow, helps to reduce coding errors and increases the flexibility of the code for future coding modifications. An explanation of each new subroutine is given in Section 2.3. An overview of how the original subroutines were broken up is given below.

BINPUT -
- INPUT_BCARDS
- INPUT_FLEX
- INPUT_JOINTS
- OUTPUT_JOINTS

DAUX-
- DAUX
- DAUX_SETUP

DAUX22
- DAUX22_SETUP

DINTG-
- DINTG
- DINTG_BACKUP
- DINTG_HALF

EJOINT-
- EJOINT
- EJOINT_TORQUE

EQUILB –
- EQUILB
- EQUILB_SOLVE
- EQUILB_SOLVE_SUB
- INPUT_EQUILB

EVALFD-
- EVALFD
- EVALFD_INTEGRAL
- EVALFD_POLY
- EVALFD_TAB

FINPUT-
- INPUT_AIRBAG_FORCE
- INPUT_BELT_FORCE
- INPUT_FCARDS
- INPUT_GLOBALGRAPHIC_FORCE
- INPUT_PLANE_FORCE
- INPUT_SEG_SEG_FORCE
- INPUT_WIND_FORCE

FRCDFL-
- FRCDFL
- FUNC_RATE_DEP

FSMSOL-
- FSMSOL
- FSMSOL_STOP

HBPLAY-
- HBPLAY
- HBPLAY_POINTS
- HPOINT_DROP

HEDING -
- HEDING
- HEDING_ACTUATORS
- HEDING_ANG_DISPL
- HEDING_BODY_PROP
- HEDING_FORCES
- HEDING_HCARDS
- HEDING_JNT_PARM
- HEDING_JOINT_FORCES
- HEDING_WATER
- HEDING_WIND

HPTURB-
- HPTURB
- HPTURB_SETUP

HSETC-
- HSETC
- HSETC_SUB

HYLPR-
- HYLPR
- HYLPR_PIVOT

INTERS-
- INTERS
- INTERS_SOLVE

KINPUT-
- INPUT_JOINT_TORQUE
- INPUT_WIND

MAIN_ATB
- INPUT_ACARDS
- INITIALIZE
- INTEGRATE_TIME
- MAIN_ATB

OUTPUT –
- INPUT_H10_CARDS
- INPUT_H11_CARDS
- INPUT_H1_H3_CARDS
- INPUT_H7_CARDS
- INPUT_HCARDS
- LUNUM_FORCES
- LUNUM_HCARDS
- OUTPUT
- OUTPUT_BODY_PROP
- OUTPUT_FORCES
- OUTPUT_H9_CARDS
- OUTPUT_HCARDS
- OUTPUT_LUNUM
- OUTPUT_SETUP

POSTPR-
- POSTPR
- READ_TAPE_8

SINPUT-
- INPUT_BELTS
- INPUT_CONSTRAINTS
- INPUT_DCARDS
- INPUT_ELLIPSOIDS
- INPUT_FORCE_TORQUE
- INPUT_PLANES
- INPUT_SPRING_DAMPERS
- INPUT_SYMMETRY

UPDATE-
- UPDATE
- UPDATE_CONSTRAINTS
- UPDATE_EULER_JOINTS
- UPDATE_JOINTS

UPDFDC-
- DEF_NEW_CUBIC
- DEF_NEW_QUADRATIC
- UPDATE_FRC_DEF_CURVE
- UPDATE_TAB

VISPR-
- VISPR
- VISPR_TORQUE

WATINP-
- INPUT_PER_FLOAT_DEV
- INPUT_WATER
- INPUT_WATER_ELLIPSOIDS
- INPUT_WATER_OUTPUT
- INPUT_WAVES
- INPUT_

WINDY-
- FORCE_TORQUE
- WINDY
- WIND_AREA
- WIND_GRID

2.2.10 Percentage Completion of Run

The percent of completion of a run is now output to the screen, as is the actual number of steps completed out of the total number of steps in a run. This provides the user with some idea of the time remaining to complete the simulation. Furthermore, by providing continuous output, the user can see whether very long runs have hung the system or are actually just taking a long time to complete.

2.2.11 Time Windowing of Output

Over time, the number of allowable plane-segment contacts, segment-segment contacts and many other types of contacts has increased substantially. Consequently, the number of potential tabular time histories has grown substantially. In addition, longer simulation times are being used for many simulations. The resultant disk space requirement for long runs, with many specified contacts, has grown enormously. Although there are several options to limit the amount of output by limiting output to specific contacts, there are times when one may want all the contact output, but only for a specific period of time during the simulation. This may occur when debugging coding changes or if a spike in a segmental acceleration shows up but the cause is hard to determine.

An option has been added to specify time windows that can limit the output from the tabular time histories, the AOU output, and the graphics data output. The user specifies a beginning time and ending time for when the data are to be printed. There can be up to three time windows

specified, which can also overlap. These time windows can be applied individually to any of the three types of output, or can apply to any combination of output. The structure OUTPUT_TIMES was added to the code to perform this function. The effect of the time windows is to limit the output that would normally be produced using the currently existing control mechanisms, such as the NPRT array from Card A.5 [10,11]. These existing features continue to function as previously, but the output they generate is limited to being printed only during the specified time windows. A new card, Card A.6, controls this output. NPRT(34) from Card A.5 flags the use of time windowing and the need for Card A.6. If NPRT(34) is blank or 0, no time windowing is desired, hence Card A.6 is not required. This allows previous inputs to be used without alteration. Only if time windowing is desired is there a need to alter the input deck.

If NPRT(34) is greater than 0, its value specifies how many time windows are desired. The current limit is 3. There must be one Card A.6 for each time window specified. The format of Card A.6 is FORMAT (A1, 1X, A1, 1X, A1, 1X, F10.3, 1X, F10.3). The input statement is READ AP, AT, AV, TS, TE, where AP is the control flag for Subroutine PRINT output, AT is the control flag for the tabular time histories and AV is the control flag for the graphics data output. An "f" or "F" in the AP, AT or AV field means that the time window does not apply to the respective type of output. A "t" or "T" means that the time window does apply to the respective type of output. TS is the start of the time window, in seconds, and TE is the end of the time window in seconds. The times are absolute times, where time 0 is the beginning of the simulation.

2.2.12 Renamed Subroutines

The following Version V.1 subroutines were renamed with longer subroutine names to clarify their purpose. No significant changes were made to these subroutines, other than the general changes that were made to all the code that are explained in detail above.

ADDMAS	WATER_ADDED_MASS
AIRBG1	INPUT_AIRBAGS
CINPUT	INPUT_FUNCTIONS
DRGCHK	WATER_DRAG_CHK
ELTIME	E_ELTIME
ETA	WAVE_HEIGHT
FILES	INPUT_FILES
FXINPT	INPUT_DEFORM
HINPUT	INPUT_HARNESS
INITAL	NPUT_INITIAL_CONDITIONS
INPROJ	INPUT_PROJANG
INTANG	INPUT_ORIENT
INTLIN	INPUT_LINEAR
LTIME	L_LTIME
ROBINP	INPUT_ROBOTICS
UPDFDC	UPDATE_FRC_DEF_CURVE
UPDPFD	UPDATE_PFD
USER	ACTUATOR_TORQUE
VINPUT	INPUT_VEHICLE
WATHED	HEDING_WATER
WATINP	INPUT_WATER
WATOUT	OUTPUT_WATER
WATSET	LUNUM_WATER
WAVEL	WATER_PNT_VELOCITY
WELFOR	WATER_ELLIP_FORCE
WFORCE	WATER_FORCE

2.2.13 Modifications Relevant to Free Form Source Code

All comments throughout the code were prefaced with a "!C" instead of a "C". In addition, all continuation lines were denoted by a "&" instead of a "*" or other character. In free source form, there are no restrictions limiting statements to specific positions on a Fortran line, the blank character is significant and may be required to separate lexical tokens, and the exclamation mark (!) is used to indicate the beginning of a comment that ends with the end of the line. If a continuation line is to follow, an "&" must be used at the point in the current line where the continuation line is to start. The rules for fixed form source code limit Fortran statements between positions 7 and 72, blanks are not significant, a "C" or "*" in position 1 indicates a comment, and any character other than a blank or 0 in position 6 indicates a continuation line.

The use of the free source form is not recommended because to the author it seems to make the code harder to read. However, it is possible to use a form of the source code that is valid and equivalent for either free source form or fixed source form. To do so requires that the following rules be followed in the source code [5, page 77].

a) Limit labels to positions 1 through 5, and statements to positions 7 through 72 – this was maintained throughout the code.
b) Treat blanks as significant – should apply throughout the code.
c) Use the exclamation mark (!) for a comment, but don't place it in position 6. Do not use C or * forms for a comment – this was done throughout the code.
d) To continue statements, use the ampersand in both position 73 of the line to be continued, and in position 6 of the continuation. Positions 74 to 80 must remain blank or have only a comment there. Positions 1 through 5 must be blank. This was followed throughout the code, except that an ampersand was not placed in position 73.

To complete the process to make the code valid and equivalent for either free source form or fixed source form requires only that all statements that are continued need an ampersand in position 73.

2.2.14 STOP Statement Messages

The use of mixed case and the use of proper grammatical structure were applied to most STOP and warning messages throughout the code. These changes make it easier to read and understand these somewhat cryptic warnings and messages.

2.2.15 Addition of Blanks

A tremendous number of blanks were inserted in the Fortran part of the statements throughout the code to make the code much easier to read. The blanks were inserted in accordance with the rules for blanks in free source format code.

2.3 New Subprograms

Provided below is a description of all the new subprograms added in Version V.3 of the ATB Model. Extensive comments are also provided in the source code of each subroutine explaining in more detail the function of the subprogram. A subprogram call and caller sequence was omitted because this information is now readily available in the source browsers commonly provided with many of the new developer environments. These source browsers also provide information on the definitions and instances of references of all variables within a source code.

Subroutine CHECK_COMMENT - This subroutine parses the input records of either the .ain or .lin standard input files to determine if the current record contains comments or input data. The "#" symbol in the first column of a record denotes that record as a comment.

Function CHECK_HIGH_VALUE - This function checks the value of the high precision value passed to it. If it is smaller than or equal to the smallest single precision value, the function returns a value of single precision 0. If it greater than the smallest allowable single precision value, the function returns the single precision equivalent of the passed double precision value. This function is used in the process of outputting double precision data as single precision values to the Unit1 output file used for simulation graphics.

Subroutine CHECK_ROT_SEQ - This subroutine is used to check the validity of a rotation sequence. An example would be the rotation sequence 1-1-3, where "1" and "3" refer to the axis about which the rotation is performed. This is an invalid sequence, since there are two sequential rotations about the same axis. The subroutine would return a value of "false" for this sequence, indicating that it is an invalid sequence.

Subroutine CHECK_ROTATION_ORDER - This subroutine checks whether the rotation order of the axes of rotation given for the initial rotation angels for the JPASS segment specified by the G.3 card specify a proper rotation.

Subroutine CONVERT_TIME - This subroutine is called by Subroutine DATE_TIME to convert the elapsed CPU time for a simulation from total seconds to seconds, minutes, hours and days.

Subroutine DATE_TIME- This subroutine uses intrinsic Fortran 90 and Fortran 95 calls to get the starting and ending times for a simulation. It also computes the elapsed CPU time for the run.

Subroutine DAUX22_SUB - This subroutine performs a subset of the computations done by Subroutine DAU22 to reduce the system matrix that is solved to compute the linear and angular accelerations and constraint forces of the simulation.

Subroutine DAUX_SETUP - This subroutine was created from Subroutine DAUX and sets up the initial values of the A & B arrays and U & V vectors in which the U1 and U2 arrays are modified by the contact and joint forces.

Subroutine DEF_NEW_CUBIC - This subroutine was created in most part from Subroutine UPDFDC of Version V.1. It defines a new cubic function when Subroutine UPDFDC detects the proper conditions in the force function.

Subroutine DEF_NEW_QUADRATIC - This subroutine was created in most part from Subroutine UPDFDC of Version V.1. It defines a new quadratic function when Subroutine UPDFDC detects the proper conditions in the force function.

Subroutine DINTG_BACKUP - This subroutine computes the backup entry point of the program integrator if H, the step size in the integrator, has been halved. It was created from Subroutine DINTG.

Subroutine DINTG_HALF - This subroutine was created from Subroutine DINTG and is called by Subroutine DINTG_BACKUP if a convergence test fails. It outputs appropriate error messages and modifies the value of H, or stops the program if the converge problem is fatal.

Subroutine EJOINT_TORQUE - This subroutine computes the torques acting on an Euler joint and adds them to the U2 array. It was created from Subroutine EJOINT.

Subroutine EQUILIB_SOLVE - This subroutine adjusts the initial input position based on the parameters supplied Cards G.2 and G.3 such that the initial normal contact forces are equal to either the supplied values or those computed by the constraint forces. It was created from Subroutine EQUILB.

Subroutine EQUILB_SOLVE_SUB - This subroutine outputs error messages associated with the attempts of Subroutine EQUILB_SOLVE to find an equilibrium solution of the body at

time zero. If the solution is successful, it sets the linear position and orientation of the variable being solved for to the initial position and orientation of that segment. It was created from Subroutine EQUILB.

Subroutine EVALFD_INTEGRAL - This subroutine computes the integral of a function from D0 to D. It was created from Subroutine EVALFD.

Subroutine EVALFD_POLY - This subroutine computes the 5^{th} order polynomial or derivative of the 5^{th} order polynomial of a function. It was created from Subroutine EVALFD.

Subroutine EVALFD_TABLE - This subroutine evaluates a tabular function. It was created from Subroutine EVALFD.

Subroutine FORCE_TORQUE - This subroutine was created in most part from Subroutine WINDY of Version V.1. It computes the forces and torques associated with force and/or torque functions supplied by the D.9 cards.

Subroutine FRAME_WINDOW - This subroutine uses QuickWin library routines, part of the Visual Fortran package, to create a simple Window style interface. The interface is a single window with a top menu bar. The menu bar has only one item "File". Under the "File" is the "Exit" command. All terminal output including error messages are written to this window.

Subroutine FSMSOL_STOP - This subroutine outputs information if the set of equations to be solved by Subroutine FSMSOL, which are stored in the C array, exceeds the maximum size of this array. It was created from Subroutine FSMSOL

Subroutine FUNC_RATE_DEP - This subroutine computes and adds rate dependent functions. It was created from Subroutine FRCDFL.

Subroutine GET_MONTH_NAME - This subroutine obtains the name of a month from its numerical sequence, e.g. given the value of 07 for a month, it returns the value "July". This subroutine is used by the new feature which prints out the starting and ending times for a simulation.

Subroutine HBPLAY_POINTS - This subroutine determines if the new NL array is different from the previous NL array. If so, it recomputes the BB elements for the points that are different. What is effectively doing is determining if the harness belt points that were in play for

the previous step still in play for the current step. If not, it removes the points that are no longer in play. It was created from Subroutine HBPLAY.

Subroutine HEDING_ACTUATORS - This subroutine prints out the headings for the actuator torque tabular time histories. It was created from Subroutine HEDING.

Subroutine HEDING_ANG_DISPL - This subroutine prints out the headings for the segment angular displacement tabular time histories. It was created from Subroutine HEDING.

Subroutine HEDING_BODY_PROP - This subroutine prints out the headings for the total body properties tabular time histories. It was created from Subroutine HEDING.

Subroutine HEDING_FORCES - This subroutine was created in most part from Subroutine HEDING of Version V.1. It prints out to the tabular time history file(s) the headings associated with the plane/segment, segment/segment, simple belts, spring-dampers, force functions, torque functions, and harness-belts.

Subroutine HEDING_HCARDS - This subroutine was created in most part from Subroutine HEDING of Version V.1. It prints out to the tabular time history file(s) the headings associated with the linear accelerations (Cards H.1), the linear velocities (Cards H.2), the linear displacements (Cards H.3), rotations (Cards H.4), angular velocities (Cards H.5), angular accelerations (Cards H.6), the joint function data (Cards H.7), the wind force data (Cards H.8), joint forces and torques (Cards H.9), total body properties (Cards H.10), and the joint actuator torque histories (Cards H.11).

Subroutine HEDING_JNT_PARM - This subroutine prints out the headings for the joint parameter data (Card H.7) tabular time histories. It was created from Subroutine HEDING.

Subroutine HEDING_JOINT_FORCES - This subroutine prints out the headings for the tabular time histories of the joint forces and torques. It was created from Subroutine HEDING.

Subroutine HEDING_WATER - This subroutine prints out the headings for the tabular time histories for the water force option parameters. It was created from Subroutine HEDING.

Subroutine HEDING_WIND - This subroutine prints out the headings for the tabular time histories associated with the wind force option. It was created from Subroutine HEDING.

Subroutine HPOINT_DROP - This subroutine is used by Subroutine HBPLAY_POINTS in the process of determining which harness belt points are currently in play. It was created from Subroutine HBPLAY.

Subroutine HPTURB_SETUP - This subroutine is called by Subroutine HPTURB in the process of perturbing (adjusting) the harness belts based on the friction between the belts, the belt material properties and how the segment on which the belt lies is moving. Amongst other functions, it sets up the C and IJK_HRN elements for the tie-points of the harness in the process of solving the C matrix. It was created from Subroutine HPTURB.

Subroutine HSETC_SUB - This subroutine is called by Subroutine HSETC in the process of setting up and solving the matrix used to compute the forces and strains within the harness belts. It was created to modularize the function of Subroutine HSETC, from which it was created.

Subroutine HYLPR_PIVOT - This subroutine is called by Subroutine HYLPR, from which it was created, to find the pivot column for the Simplex method used by Subroutine HYLPR.

Subroutine INITIALIZE - This subroutine performs all the initialization and setup needed for a simulation prior to the starting the integration forward in time. It was created from the main program subprogram, MAIN_ATB.

Subroutine INPUT_ACARDS - This subroutine was created primarily from .MAIN of Version V.1. It reads in all the A Card data and writes it to the .AOU file.

Subroutine INPUT_AIRBAG_FORCE - This subroutine reads in the force definitions for the airbags from the F.6 cards. It was created from Subroutine FINPUT.

Subroutine INPUT_BCARDS - This subroutine was created primarily from Subroutine BINPUT of Version V.1. It reads in and echoes to the .AOU file the segmental parameters associated with Cards B.1, B.2 and B.6.

Subroutine INPUT_BELT_FORCE - This subroutine reads in the force definitions for the simple belt force and contact ellipsoid interaction, represented by the F.2 cards. It was created from Subroutine FINPUT.

Subroutine INPUT_BELTS - This subroutine controls the reading in and echoing of the input cards that describe the physical dimensions of the simple restraint belts (Cards D.3). It was created from Subroutine SINPUT.

Subroutine INPUT_CONSTRAINTS - This subroutine reads in and echoes the parameters describing the various types of constraints that can be specified for the segments, as provided by Cards D.6. It was created from Subroutine SINPUT.

Subroutine INPUT_DCARDS - This subroutine controls the subroutines that read and echo the input cards that describe the physical dimensions of the contact planes, simple belts, contact ellipsoids, body segment symmetry and spring dampers. It replaces Subroutine SINPUT in function, from which it was derived.

Subroutine INPUT_ELLIPSOIDS – this subroutine reads in and echoes the input parameters describing the additional contact ellipsoids, as well as altering the values for the default contact ellipsoids implicitly associated with each segment, as provided by the D.5 cards. It was created from Subroutine SINPUT.

Subroutine INPUT_EQUILB - This subroutine reads in the parameters associated with placing a simulation body in an equilibrium position before the start of the integration forward in time, as provided by Cards G.4, G.5 and G.6. It was created from Subroutine EQUILB.

Subroutine INPUT_FCARDS - This subroutine controls the subroutines which input the parameters associated with the plane-segment, segment-segment, simple belt-segment, harness belt-segment, airbag-segment, wind force and water force contacts and globalgraphic joint forces. It replaces Subroutine FINPUT, from which it was created.

Subroutine INPUT_FLEX – This subroutine was created primarily from Subroutine BINPUT of Version V.1. It reads in and echoes the flexible element parameters associated with Card B.7.

Subroutine INPUT_FORCE_TORQUE - This subroutine reads in the parameters controlling the force / torque functions as supplied by the D.9 cards. It was created from Subroutine SINPUT.

Subroutine INPUT_GLOBALGRAPHIC_FORCE - This subroutine reads in the parameters that specify the allowed contacts for the globalgraphic joints, as provided by the F.4

cards. It also sets up tables to control the time history information for each function for the globalgraphic joints. It was created from Subroutine FINPUT.

Subroutine INPUT_H10_CARDS - This subroutine reads in and echoes the parameters that control the output of the tabular time histories for body center of gravity and related information, as specified by the H.10 cards. It was created from Subroutine OUTPUT.

Subroutine INPUT_H11_CARDS - This subroutine reads in and echoes the parameters that control the output of the tabular time histories for the joint actuator torques, as specified by the H.11 cards. It was created from Subroutine OUTPUT.

Subroutine INPUT_H1_H3_CARDS - This subroutine reads in and echoes the parameters that control the output of the tabular time histories for the total accelerations of a point on a segment, the relative velocities of a specified point on a segment, and relative linear displacements of a point on segment, as specified by Cards H.1, H.2 and H.3. It was created from Subroutine OUTPUT.

Subroutine INPUT_H4_H9_CARDS - This subroutine reads in and echoes the parameters that control the output of the tabular time histories for the segment angular accelerations, segment relative angular velocities, relative angular displacements, wind forces and joint forces and torques, as specified by Cards H.4, H.5, H.6, H.8, H.9, respectively. It was created from Subroutine OUTPUT.

Subroutine INPUT_H7_CARDS - This subroutine reads in and echoes the parameters that control the output of the tabular time histories for the joint parameters, as specified by Cards H.7. It was created from Subroutine OUTPUT.

Subroutine INPUT_HCARDS - This subroutine was created primarily from Subroutine OUTPUT of Version V.1. It controls the subroutines that read in and echo the input parameters that specify the tabular time histories associated with Cards H.1 through Cards H.11.

Subroutine INPUT_JOINT_TORQUE- This subroutine reads in and echoes the parameters associated with the joint restoring force and torque functions, as specified by the E.7 cards. It was created from Subroutine KINPUT.

Subroutine INPUT_JOINTS - This subroutine was created primarily from Subroutine BINPUT of Version V.1. It reads in the joint parameters associated with Cards B.3, B.4 and B.5.

Subroutine INPUT_PER_FLOAT_DEV - This subroutine reads in and echoes the parameters associated with water force personal flotation devices, as specified by Cards F.9.h – F.9.j. It was created from Subroutine WATINP.

Subroutine INPUT_PLANE_FORCE - This subroutine reads in and echoes the parameters associated with the plane – segment contact forces, as specified by Cards F.1. It was created from Subroutine FINPUT.

Subroutine INPUT_PLANES - This subroutine reads in and echoes the parameters defining the plane – segment contact forces, as specified by Cards D.2. It was created from Subroutine SINPUT.

Subroutine INPUT_SEG_SEG_FORCE - This subroutine reads in and echoes the parameters specifying the allowed segment – segment contacts (more precisely, contact ellipsoid – contact ellipsoid), as specified by Cards F.3. It was created from Subroutine FINPUT.

Subroutine INPUT_SPRING_DAMPERS - This subroutine reads in and echoes the parameters defining the spring dampers that can be connected between segments, as specified by Cards D.8. It was created from Subroutine SINPUT.

Subroutine INPUT_SYMMETRY - This subroutine reads in and echoes the parameters setting which body segment symmetries are desired, as specified by the D.7 cards. It was created from Subroutine SINPUT.

Subroutine INPUT_WATER_ELLIPSOIDS - This subroutine reads in and echoes the parameters associated with the water force ellipsoid data, as specified by Cards F.9.e – F.9.g. It was created from Subroutine WATINP.

Subroutine INPUT_WATER_OUTPUT - This subroutine reads in and echoes the parameters associated with the water force output data, as specified by Cards F.9.k – F.9.m. It was created from Subroutine WATINP.

Subroutine INPUT_WAVES - This subroutine reads in and echoes the parameters associated with the water force waves option, as specified by Cards F.9.b – F.9.d. It was created from Subroutine WATINP.

Subroutine INPUT_WIND - This subroutine reads in the parameters defining the wind force functions and drag coefficients, as specified by Cards E.6. It was created from Subroutine KINPUT.

Subroutine INPUT_WIND_FORCE - This subroutine reads in the parameters to set up the tables that control the tabular time histories for each allowed wind force contact, as specified by Cards F.7.a – F.7.c. It was created in part from Subroutine FINPUT, but was greatly expanded in function.

Subroutine INTERS_SOLVE - This subroutine is used by Subroutine INTERS, from which it was created, in the process of determining the intersection of two contact ellipsoids. One of the functions of Subroutine INTERS_SOLVE is to test for the convergence of the iterative process of determining the intersection.

Subroutine LUNUM_FORCES -This subroutine computes the logical unit numbers associated with the tabular time histories for the force data output. This includes the simple belt forces, the harness belt forces, the spring damper forces, the plane/segment contact force data, the segment/segment contact force data and other force data output. Tests are performed to ensure that the maximum number of logical units allowed to be open concurrently is not exceeded.

Subroutine LUNUM_HCARDS - This subroutine computes the logical unit numbers associated with the tabular time histories for the output associated with Cards H.1 through H.10. Tests are performed to ensure that the maximum number of logical units allowed to be open concurrently is not exceeded.

Module MODULE_FLEXIBLE - This module contains all the global variables associated with the deformable body option. Refer to Section 2.1.1.b for more details. A listing of this module is contained in Appendix A.2.

Module MODULE_STANDARD - This module contains all the global variables associated with the ATB Model prior to the addition of the deformable body and water force options. It also contains all the ATB Model global parameters. Refer to Section 2.1.1.a for more details. A listing of this module is contained in the Appendix A.1.

Module MODULE_WATER - This module contains all the global variables associated with the water force option. Refer to Section 2.1.1.c for more details. A listing of this module is contained in the Appendix A.3.

Subroutine OUTPUT_BODY_PROP - This subroutine was created primarily from Subroutine OUTPUT of Version V.1. It outputs the tabular time history data associated with the total body properties specified by Cards H.10.

Subroutine OUTPUT_FORCES - This subroutine outputs the various force tabular time histories to either the *.tp8 temporary file or to the individual tabular time history files. It was created in most part from Subroutine OUTPUT of Version V.1. It is called only by Subroutine OUTPUT.

Subroutine OUTPUT_H9_CARDS - This subroutine was created primarily from Subroutine OUTPUT of Version V.1. It outputs the tabular time history data associated with the joint constraint forces and torques for the joints specified by Cards H.9.

Subroutine OUTPUT_HCARDS - This subroutine was created primarily from Subroutine OUTPUT of Version V.1. It controls the printing of output associated with Cards H.1 through H.8.

Subroutine OUTPUT_JOINTS - This subroutine was created primarily from Subroutine BINPUT of Version V.1. It echoes the joint parameters associated with Cards B.3, B.4 and B.5.

Subroutine OUTPUT_LUNUM - This subroutine outputs to the .AOU file the names of the files opened and/or used for the simulation. It also lists which type of data are in the tabular time history files that were opened. It is called only by Subroutine OUTPUT_SETUP.

Subroutine OUTPUT_SETUP - This subroutine sets up the files for output of the tabular time histories, Tape 8 output, or both. It acts as an executive subroutine to perform many of the functions that had been carried out by Subroutine OUTPUT of Version V.1. Accordingly, it was created in most part from Subroutine OUTPUT of Version V.1. It is called only by Subroutine OUTPUT.

Subroutine READ_TAPE_8 - This subroutine was created primarily from Subroutine POSTPR of Version V.1. It reads in the simulation data that were stored in the unformatted Tape8 temporary file. It is called only by Subroutine POSTPR.

Subroutine UPDATE_CONSTRAINTS - This subroutine updates the constraints, as defined by the D.5 cards. It was created from Subroutine UPDATE.

Subroutine UPDATE_EULER_JOINTS - This subroutine test whether to lock or unlock an Euler joint axis. It applies the same test as in Subroutine UPDATE_JOINTS, but to each axis of the Euler joint separately. It was created from Subroutine UPDATE.

Subroutine UPDATE_JOINTS - This subroutine checks for an impulse on a joint stop to determine if the state of a joint should remain locked, or if the joint should unlock. It was created from Subroutine UPDATE.

Subroutine UPDATE_TAB -This subroutine was created primarily from Subroutine UPDFDC of Version V.1. It updates many of the parameters that are stored in the TAB array. This array holds the constants associated with the various force functions used by the program.

Subroutine VISPR_TORQUE - This subroutine computes the total torque for the joints in the inertial reference system, then converts it to the local reference system and adds it to the external angular acceleration array (EXT_ANG_ACL, was U2). It was created from Subroutine VISPR.

Subroutine WIND_AREA -This subroutine computes the projected area subjected to the wind forces. It was created primarily from Subroutine WINDY.

Subroutine WIND_GRID - This subroutine computes the grid used to discretize the application of the wind forces. It was created primarily from Subroutine WINDY.

2.4 Significantly Altered Subroutines

The following subroutines were significantly altered, usually by breaking them up by function into several smaller subroutines.

Subroutine HEDING - Its function of outputting the heading data for the tabular time histories is now distributed between Subroutines HEDING_FORCES and HEDING_HCARDS. The revised Subroutine HEDING serves as the controlling subroutine, calling Subroutines HEDING_FORCES and HEDING_HCARDS.

Subroutine INPUT_FILES – Though it is listed above as being the rename of Subroutine FILES, this subroutine has been extensively rewritten to query the user for the default directory name where the input files are to be found and the output files are to be stored. It updates a parameters file that saves the directory that was used when the program last ran. It asks the user for the name of the input file and the name to be given to the output files of the simulation.

Subroutine POSTPR - Its function of outputting tabular time histories and controlling the computing of injury criteria remains unchanged. However, the task of reading in the data stored in the temporary file Tape8 has been removed and placed in Subroutine READ_TAPE_8.

Subroutine OUTPUT - Its function of both inputting the H.1 through H.11 cards and outputting all the tabular time histories, except for the water forces, is now distributed among

many subroutines. The revised Subroutine OUTPUT serves as the controlling subroutine, which calls the following subroutines: INPUT_HCARDS, OUTPUT_BODY_PROP, OUTPUT_FORCES, OUTPUT_H9_CARDS, OUTPUT_HCARDS, and OUTPUT_SETUP.

Subroutine UPDFDC - Its function of updating the force functions has been partitioned into the following subroutines: DEF_NEW_CUBIC, DEF_NEW_QUADRATIC, UPDATE_TAB.

Subroutine WATINP - its function of setting up and controlling the water force option has been partitioned into the following subroutines: INPUT_PER_FLOAT_DEV, INPUT_WATER, INPUT_WATER_ELLIPSOIDS, INPUT_WATER_OUTPUT, and INPUT_WAVES.

Subroutine WINDY - Its functions of computing the wind forces and the forces associated with the force/torque functions, are now separated by function. The calculation of the force/torque forces is now in Subroutine FORCE_TORQUE. The computation of the wind forces is now controlled by the revised Subroutine WINDY, which calls Subroutines WIND_AREA and WIND_GRID.

2.5 Eliminated Subroutines

The following subprograms found in Version V.1 of the ATB Model have been eliminated from Version V.3:

Subroutine BINPUT – Its function of inputting all the B cards has been replaced by the following new subroutines: INPUT_BCARDS, INPUT_FLEX, INPUT_JOINTS, and OUTPUT_JOINTS.

Subroutine BLOCKDTA - The BLOCK DATA subprogram has been replaced by the use of three modules: MODULE_STANDARD, MODULE_FLEXIBLE, and MODULE_WATER..

Subroutine COMPSYS - Its function of specifying the computer system on which the ATB Model is running has been incorporated in MODULE_STANDARD.

Subroutine ELAREA - Its function of evaluating a function over the area of an ellipse has been replaced by Subroutine ELARE3.

Subroutine FINPUT – Its function of inputting the F cards has been replaced by the following subroutines: INPUT_AIRBAG_FORCE, INPUT_BELT_FORCE, INPUT_FCARDS, INPUT_GLOBALGRAPHIC_FORCE, INPUT_PLANE_FORCE, INPUT_SEG_SEG_FORCE, INPUT_WIND_FORCE

Subroutine KINPUT – Its function of inputting all the B cards has been replaced by the following new subroutines: INPUT_JOINT_TORQUE and INPUT_WIND

Subroutine SINPUT – Its function of inputting all the D cards has been replaced by the following new subroutines: INPUT_BELTS, INPUT_CONSTRAINTS, INPUT_DCARDS, INPUT_ELLIPSOIDS, INPUT_FORCE_TORQUE, INPUT_PLANES, INPUT_SPRING_DAMMERS, INPUT_SYMMETRY.

Function SPRNGF - Its function of computing the nonlinear spring torque for the joints has been performed by other joint subroutines for several previous versions of the model.

Function VECANG - Its function of computing the angle between two vectors is no longer needed by the program.

2.6 List-Directed ATB Input File

A new type of ATB input file based on the list-directed sequential READ statement is provided to solve various limitations associated with the fixed format input file, i.e., the .AIN input file. It uses "LIN" as the file extension and is essentially a free format version of the .AIN file.

The .LIN input file has the same modular input deck style as the .AIN file. In most cases, the variable list for each card is the same between the two. A .LIN input card simply stretches each numerical value in its corresponding .AIN card to more than 10 digits and separates the values with a blank space as the delimiter. Character strings usually requires double quotation marks and are also blank space delimited. In addition, users have the option to add one or more comments at the beginning of each input block by including records with "#" in the first column of the record for either the .LIN file or the .AIN file.

For some cards, such as Card H.4, the .AIN file has a second card if the number of input values exceeds 11 in the first card. In the .LIN file, there is only one card with the number of input values limited by the allowable record length set the by operating system. The cards having this change are F.7.b, H.4 to H.9, and H.11. All the other cards have the same number of input values for both the .AIN and .LIN files.

With free format, the user can edit input decks much faster and easier in a .LIN file. To help convert existing .AIN files into .LIN files, the option is provided to read in the existing .AIN file and write out the corresponding .LIN file automatically.

2.7 Miscellaneous Modifications

2.7.1 Addition of Simple QuickWin Interface

During the testing of the new ATB V.3 program, it was found that as a COMPAQ Visual Fortran console application its running window tends to close immediately upon the termination of the program. This behavior is caused by the COMPAQ Visual Fortran compiler and makes reading any program generated error messages very difficult because most of them are terminal output and will disappear after the running window closes.

To address this problem, QuickWin library routines, part of the Visual Fortran package, were used to create a simple Window style interface. The interface is a single window with a top menu bar. The menu bar has only one item "File". Under the "File" is the "Exit" command. All terminal output including error messages are written to this window. After the program stops due to an either successful execution or abnormal crash, a message box appears to ask the user whether or not to exit window. If the user chooses not to exit, the window stays open and he/she can review all the terminal output by moving the vertical scrollbar.

2.7.2 Increase the Maximum Number of Time Histories

The maximum number of time history files is 65 in version V.1. This is primarily limited by the old DOS environment in which version V.1 runs. Because the new ATB V.3 does not run in a DOS environment, the maximum number of time histories has been increased to 200. It can be increased further and up to the limit set by the computer operating system in which the program is being run.

Since each time history is assigned a specific file extension (starting with "t21") as its identifier, and the maximum number of time histories has been increased from 65 to 200, subroutine FNAME has been modified to handle file extensions of 4 characters, such as "t123".

3. VERIFICATION SIMULATIONS

A number of verification simulations were performed to compare the results generated by Version V.1 and Version V.3. Most of the input decks were provided by the Air Force. Several additional input decks were created to isolate specific problems. All simulations were run on a Dell Precision 220 Workstation (400 MHz Pentium) computer running Windows NT 4.0, service pack 5, with the code compiled with the COMPAQ Visual Fortran Compiler, Version 6.5. Most of the simulations were also repeated on a 233 MHz Dell Pentium. Several simulations were also repeated on a Silicon Graphics Indigo 2 (R10000) workstation where the code was compiled using the Silicon Graphics MIPSpro F90 compiler.

In most cases, the resultant acceleration of a point on a segment was used to compare the two versions of the code. This parameter was chosen because the point acceleration on a segment, such as the head, is usually very sensitive to the external forces imposed on the body, such as the harness belts or interaction with the contact planes. The model actually solves for the linear and angular accelerations and constraint forces, therefore it was thought advisable to compare one of the variables that the model integrator solves for. Where appropriate, other parameters were used for comparison and the parameter used is noted in the data set description.

3.1 Simulation Data Sets and Results

A brief summary of each input data set used in the verification simulations is provided below. A listing of the first part of the .AOU file (that part echoing the .AIN file) for most of the simulations is provided in the Appendices. Several .AOU files were omitted because of their length.

3.1.1 Ball Data Set

A single spherical ball bouncing on a horizontal plane is modeled. There is no vehicle motion. The ball has no initial velocity. It is held above the plane and released. It then falls and hits the plane and continues to bounce. This input deck was created specifically for this effort. A partial .AOU file listing is provided in Appendix C.1. The simulation was run for 2000 steps for a total simulation time of 4 seconds. The resultant acceleration of the center of the ball (point 0,0,0) is plotted in Figure 1. It is seen that both versions of the code produce essentially identical results.

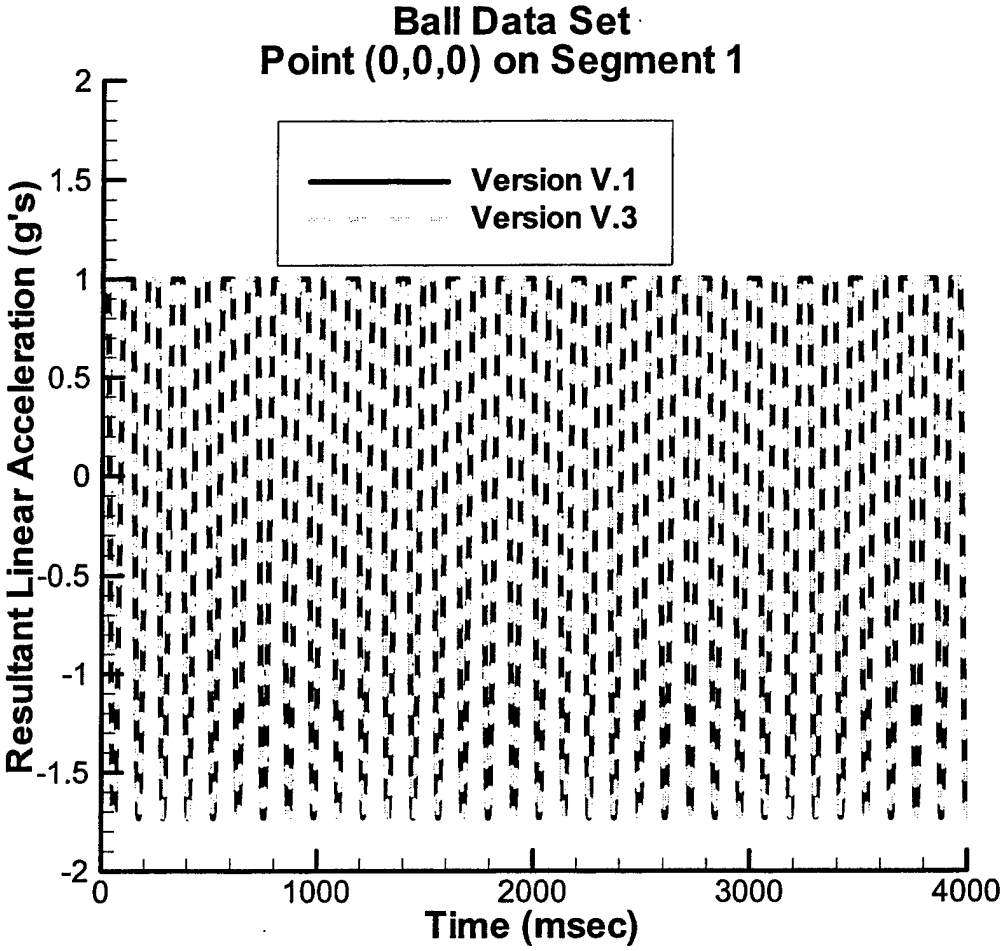

Figure 1 Ball Data Set Resultant Acceleration vs Time

3.1.2 Body Data Set

A single body composed of 15 segments and 14 joints is modeled. There is no vehicle motion and no initial linear or angular velocity. The body just falls in space. This input deck was supplied by the Air Force. A partial .AOU file listing is provided in Appendix C.2. The simulation was run for 400 steps for a total simulation time of 2 seconds. Because the intent of this data set is to look at the total body properties, the total angular momentum of the body was used as the comparison variable. As can be seen in Figure 2, the results for both versions are essentially identical.

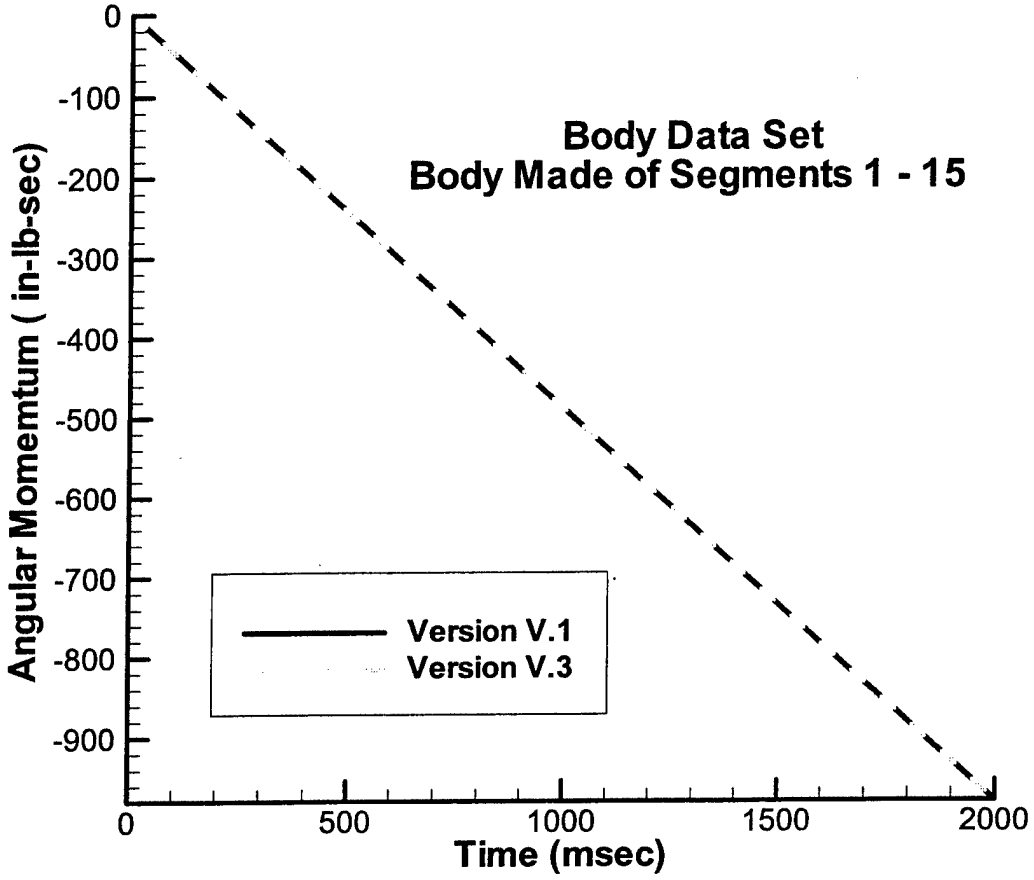

Figure 2 Body Data Set Angular Momentum vs Time

3.1.3 Ejection Data Set

A single body, composed of 16 segments and 15 joints, and an ejection seat are modeled. The vehicle models an ACES II ejection seat profile. The force environment consists of 16 contact planes, 4 additional ellipsoids, 1 harness, and a wind force. This input deck was supplied by the Air Force. No .AOU file listing is provided in the Appendices because of the length of the .AOU file. The simulation was run for 250 steps for a total simulation time of 500 milliseconds. The resultant acceleration of the center of the upper torso was used as the comparison variable. As can be seen from Figure 3, the results for both versions of the code are almost identical, with only very minor differences at some of the peaks and valleys.

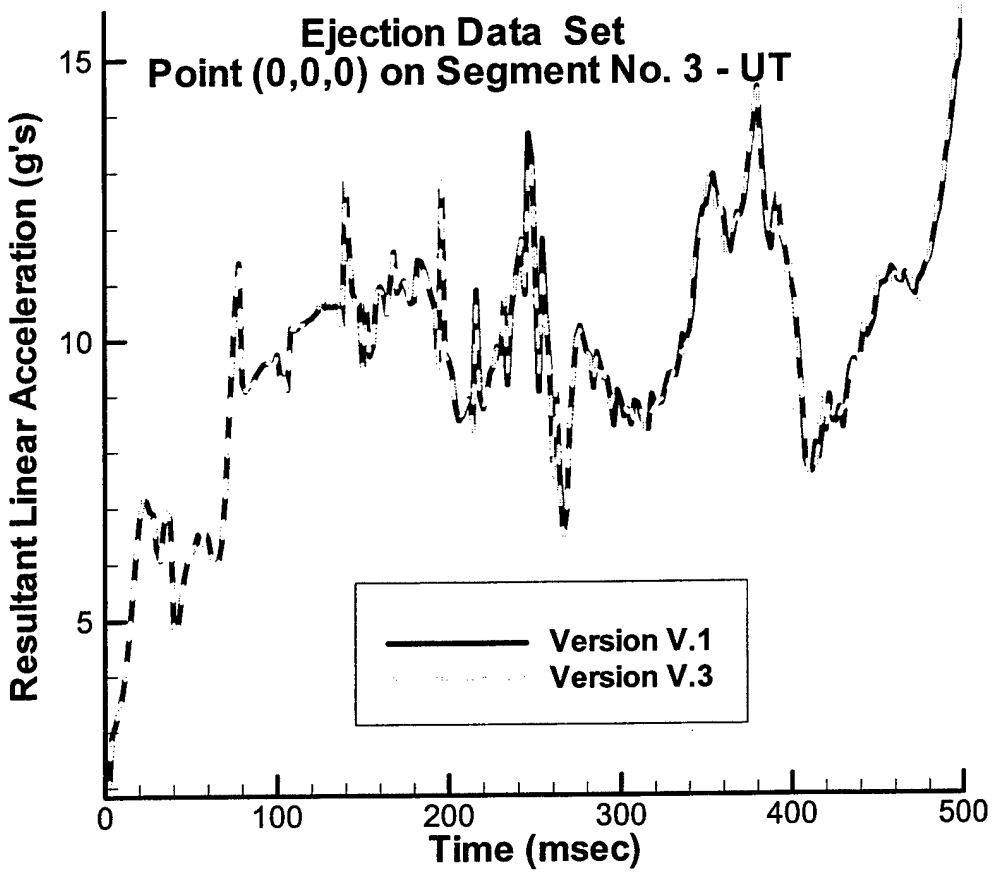

Figure 3 Ejection Data Set Resultant Acceleration vs Time

3.1.4 H4 Card Data Set

A single body composed of 3 segments and 2 joints is modeled. The vehicle models a 10 g sled impact test. There are no contact planes. The intent of this simulation is to model the head and neck responses. This input deck was provided by the Air Force. A partial .AOU file listing is provided in Appendix C.3. The simulation was run for 150 steps for a total simulation time of 300 milliseconds. The resultant acceleration of the center of the third segment was used as the comparison variable. As can be seen from Figure 4, the results for both versions of the code are essentially identical.

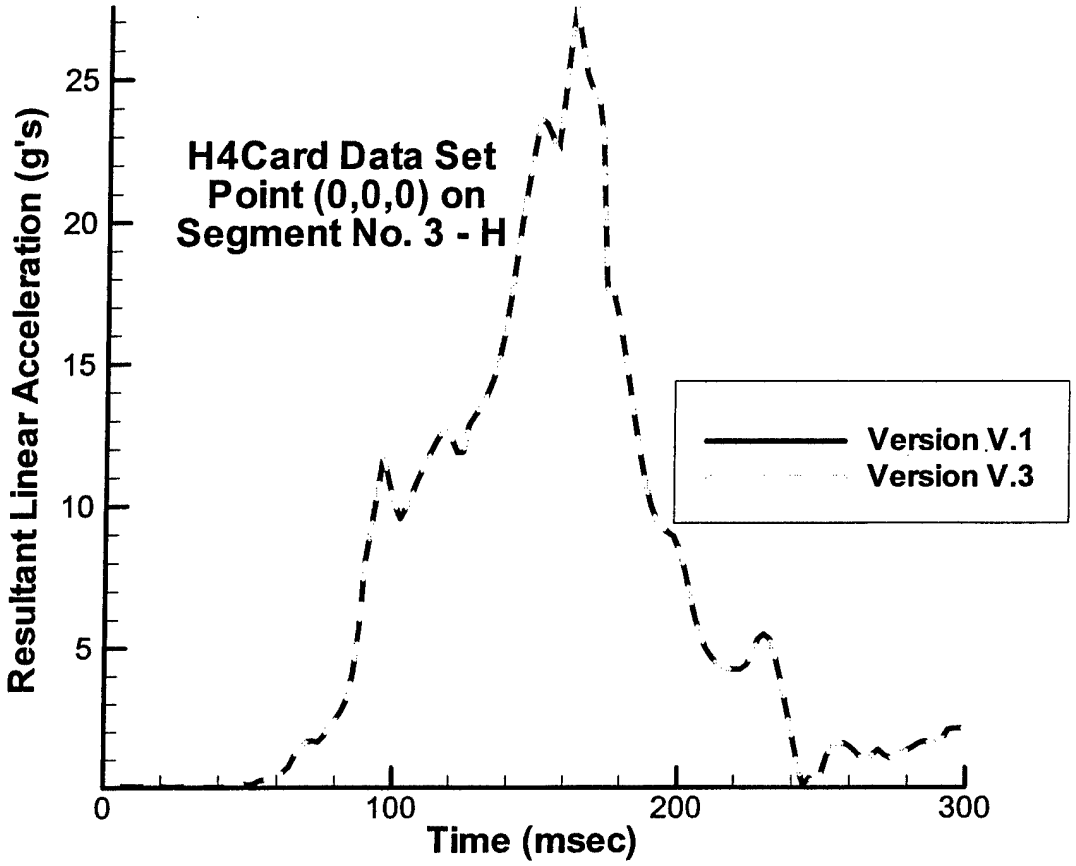

Figure 4 H4 Card Data Set Resultant Acceleration vs Time

3.1.5 Human Data Set

A single body composed of 15 segments and 14 joints is modeled. The vehicle simulates a human volunteer sled test. The joints for the body use the newly developed human joint characteristics. This input deck was provided by the Air Force. A partial .AOU file listing is provided in Appendix C.4. The simulation was run for 150 steps for a total simulation time of 300 milliseconds. The resultant acceleration of a point off-center on the head was used as the comparison variable. As can be seen from Figure 5, the results for both versions of the code are almost identical, with only extremely minor differences exhibited for some of the peaks and valleys.

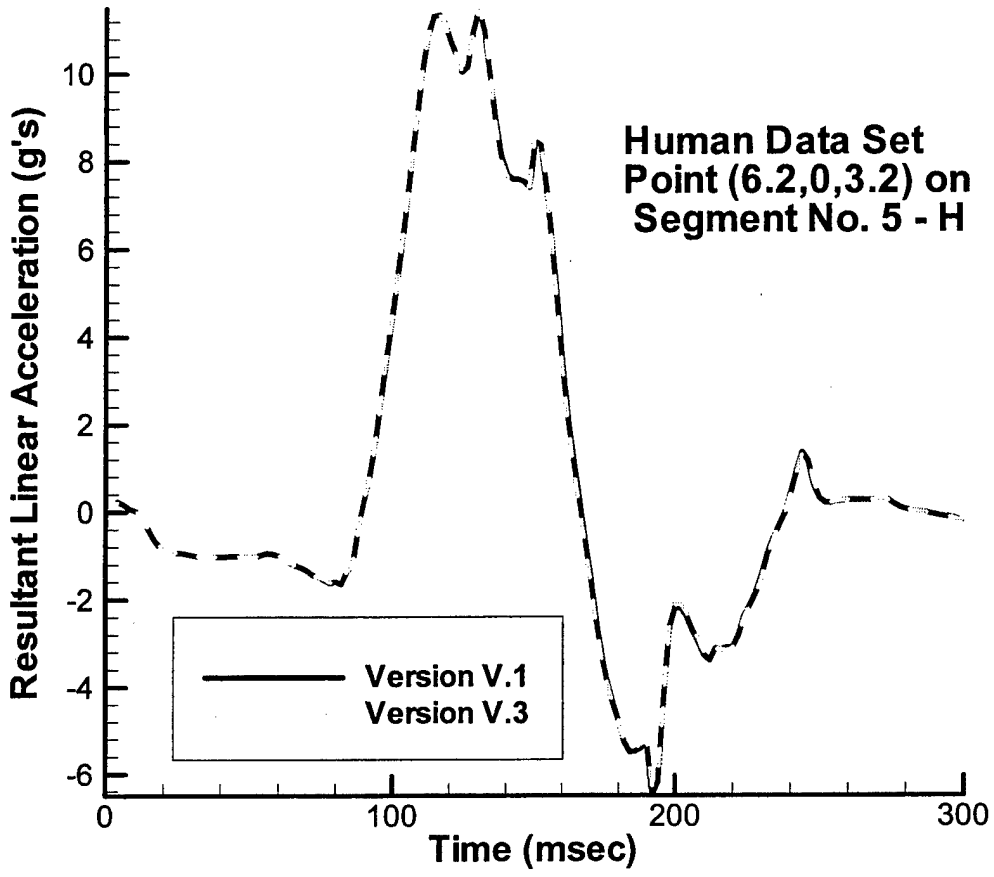

Figure 5 Human Data Set Resultant Acceleration vs Time

3.1.6 HYBIII Data Set

Two Hybrid III dummies totaling 34 segments and 33 joints are modeled. The vehicle simulates a sled test car. The force environment consists of 10 contact planes and 2 harnesses (one for each dummy). This input deck was provided by the Air Force. No .AOU file listing is provided in the Appendices because of the length of the .AOU file. The simulation was run for 400 steps for a total simulation time of 2 seconds. The resultant accelerations of the centers of the upper torso and head are used as the comparison variables. As can be seen from Figures 6 and 7, the overall responses are similar for both versions of the code, but there are significant differences throughout the simulation. When the harness belt points were not allowed to slip, as they were in the original data set, the results for both segments for both versions of the code are essentially identical, as shown in Figures 8 and 9. This result will be discussed in more detail in Section 3.4.

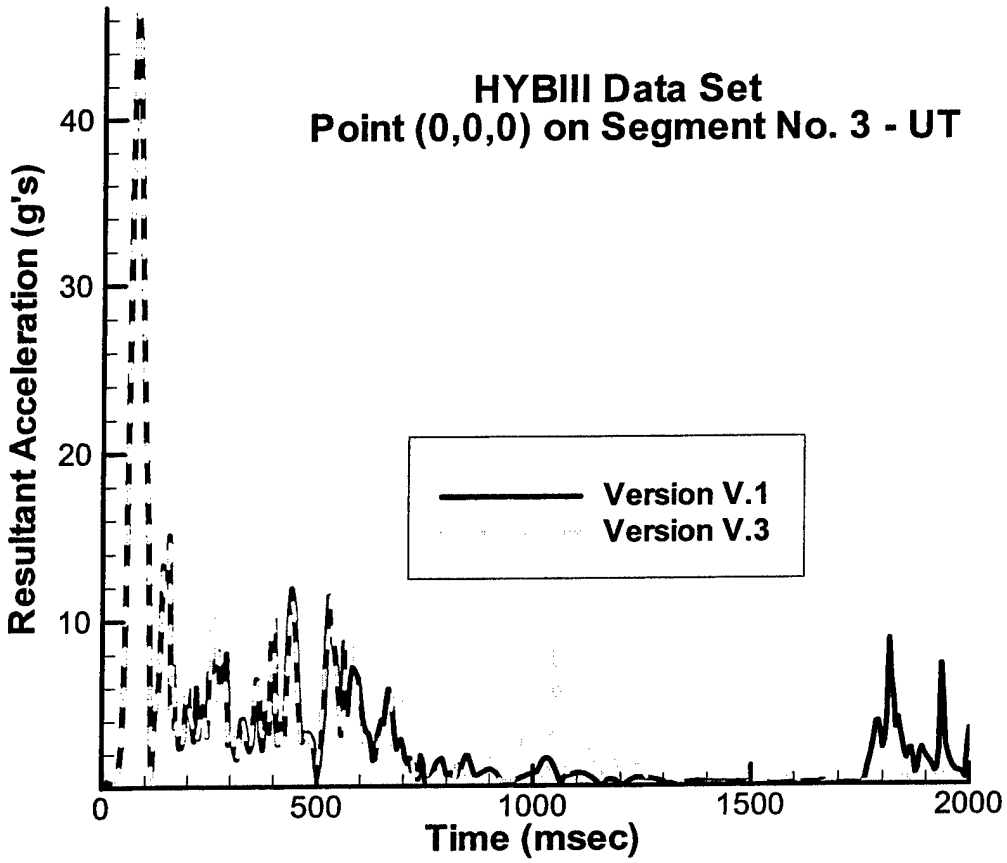

Figure 6 HYBIII Data Set Upper Torso Resultant Acceleration vs Time - Harness Belts with Sliding

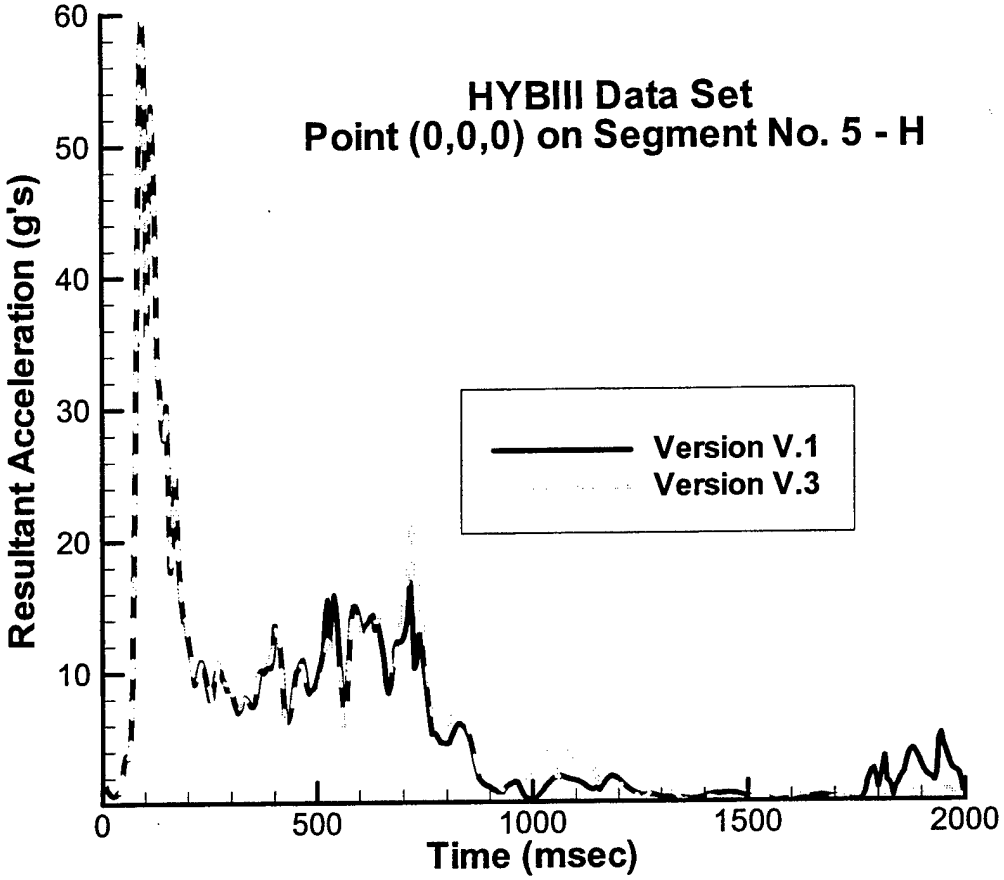

Figure 7 HYBIII Data Set Head Resultant Acceleration vs Time – Harness Belts with Sliding

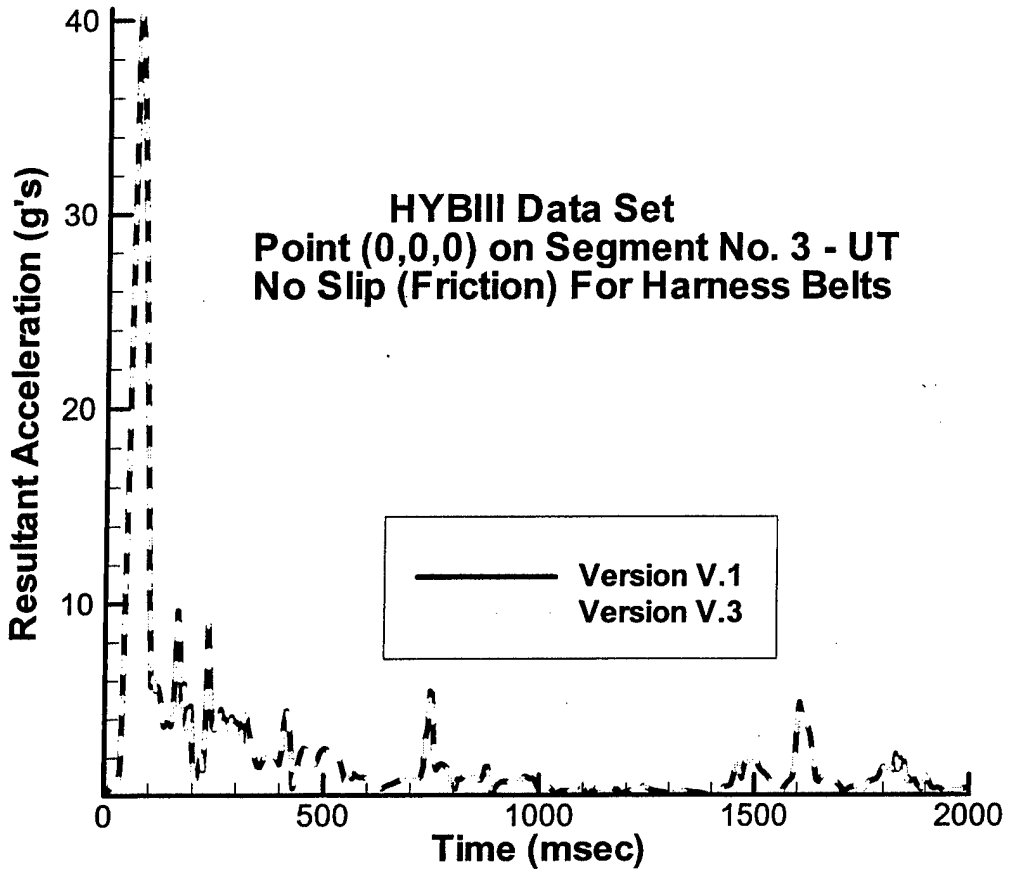

Figure 8 HYBIII Data Set Upper Torso Acceleration vs Time - Harness Belts without Sliding

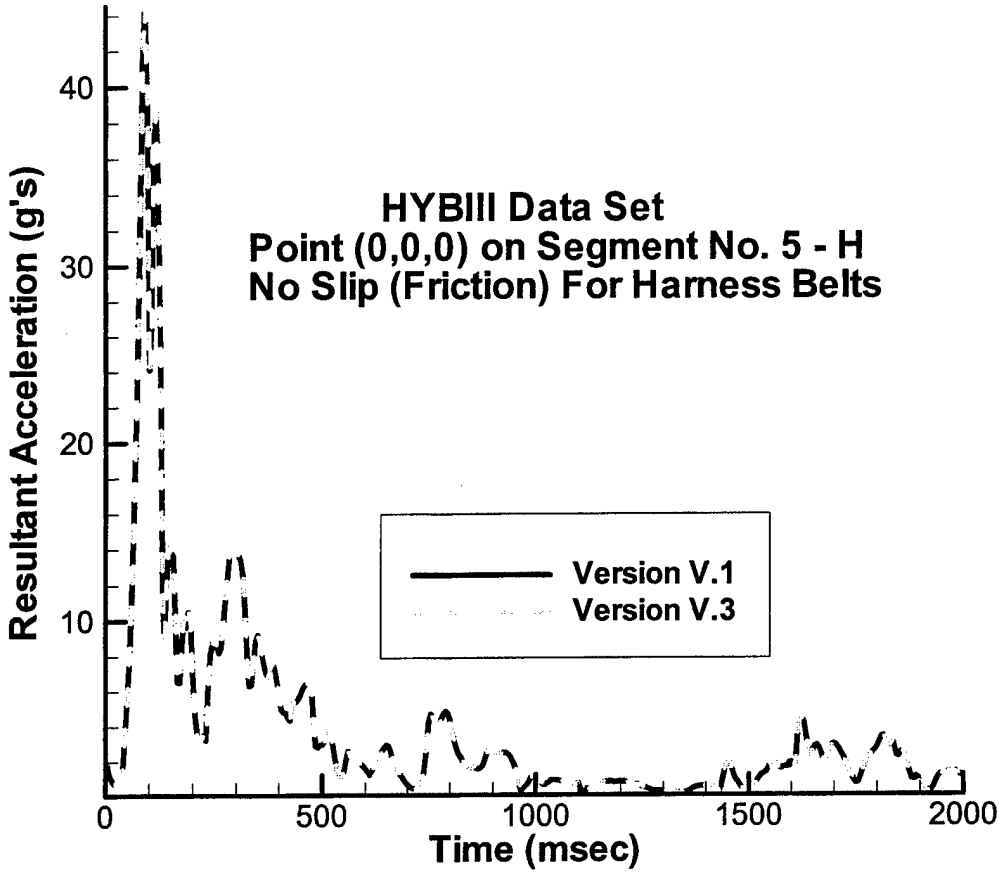

Figure 9 HYBIII Data Set Head Resultant Acceleration vs Time - Harness Belts without Sliding

3.1.7 Jump Data Set

A single body composed of 15 segments and 14 joints is modeled. There is no vehicle motion. The ground is modeled as 3 contact planes. The body is given an initial linear velocity and no initial angular velocity. The intent of the simulation is to model a parachute landing fall. This input deck was provided by the Air Force. A partial .AOU file listing is provided in Appendix C.5. The simulation was run for 80 steps for a total simulation time of 160 milliseconds. The resultant acceleration of the center of the upper torso was chosen as the comparison variable. It can be seen from Figure 10 that both versions of the code give essentially identical results.

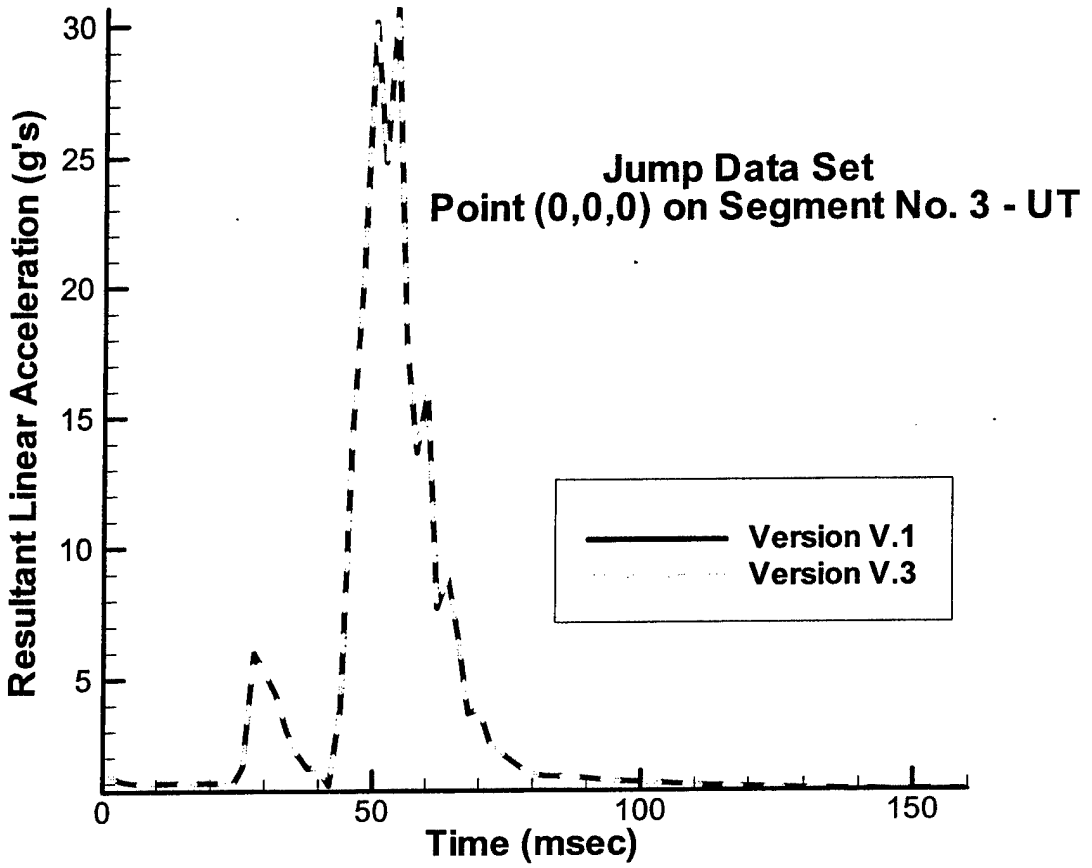

Figure 10 Jump Data Set Resultant Acceleration vs Time

3.1.8 Joint Star Data Set

A single body composed of 16 segments and 15 joints is modeled. The 16th segment represents the parachute. The body is given the initial velocity of the plane and wind forces act on the chute. This input deck was provided by the Air Force. No .AOU file listing is provided in the Appendices because of space limitations. The simulation was run for 500 steps for a total simulation time of 1 second. The resultant acceleration of the center of the neck segment was chosen as the comparison variable. As can be seen from Figure 11, the results for both versions of the code are very similar, with only very minor differences exhibited at some of the peaks and valleys.

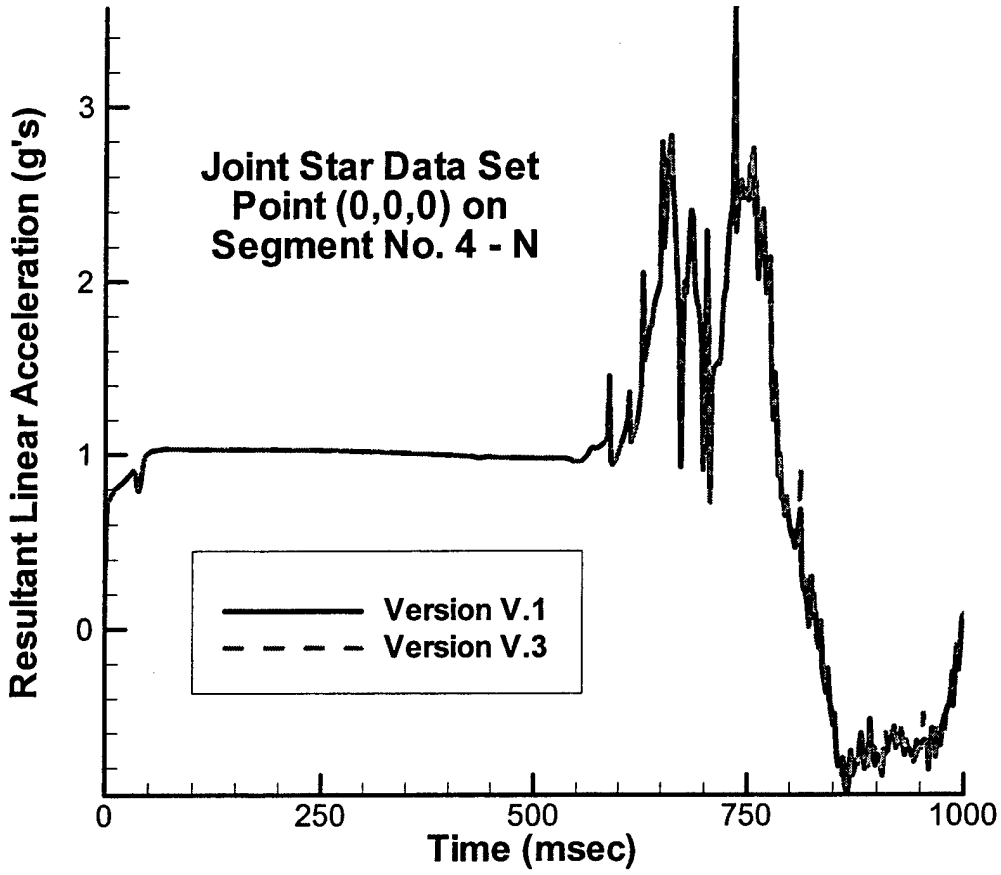

Figure 11 Joint Star Data Set Resultant Acceleration vs Time

3.1.9 Merlin Data Set

A single body composed of 7 segments and 6 joints is modeled. There is no vehicle motion. The first body segment was designated as a vehicle. The body functions as a robot using the actuator option on Card D.1.b. This input deck was provided by the Air Force. A partial .AOU file listing is provided in Appendix C.6. The simulation was run for 60 steps for a total simulation time of 600 milliseconds. Since the intent of this data set is to exercise the actuator option of the ATB Model, the Y relative angular acceleration was chosen as the comparison variable. As can be seen from Figure 12, both versions of the code produce essentially identical results.

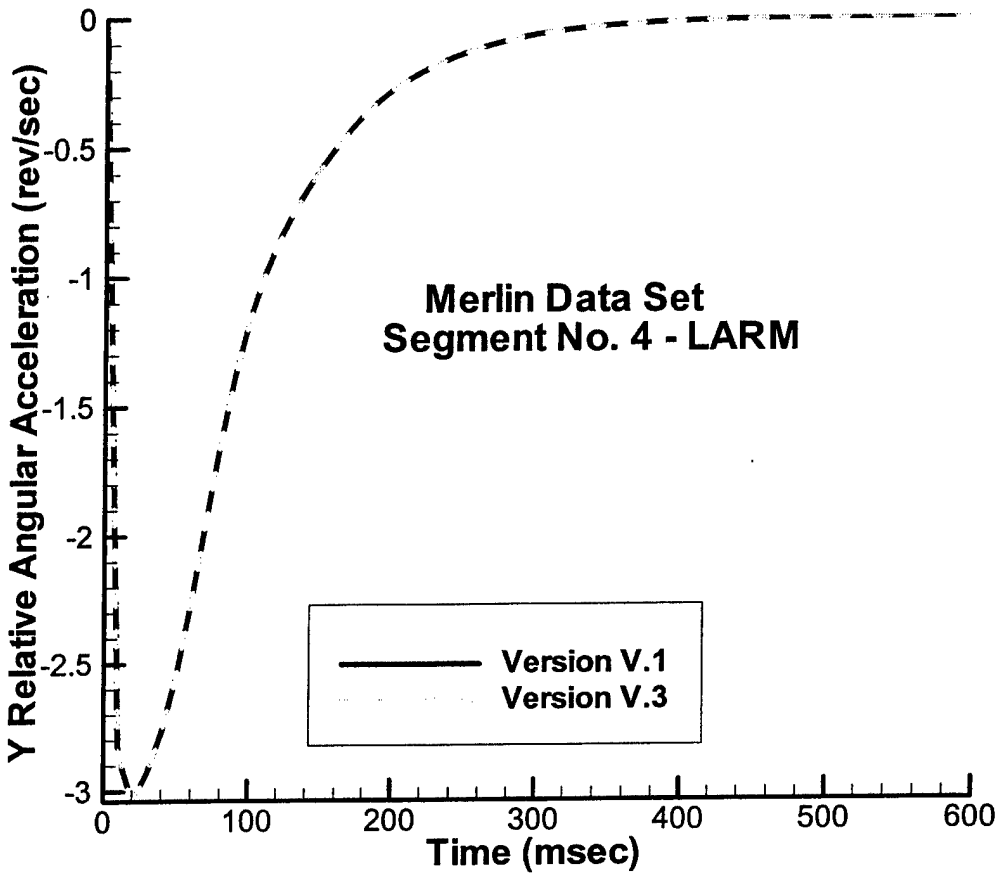

Figure 12 Merlin Data Set Y Relative Angular Acceleration vs Time

3.1.10 Neck Data Set

A single body composed of 3 segments and 2 joints is modeled. Segment 2 is modeled as a deformable segment. The vehicle models a 10 G_x deceleration. There are no contact planes. The body has no initial linear or angular velocities. This input deck was provided by the Air Force. No .AOU file listing is provided in the Appendices because of the length of the .AOU file. The simulation was run for 200 steps for a total simulation time of 400 milliseconds. The acceleration of the center of the head was used as the comparison variable. As can be seen from Figure 13, the results for both versions of the program are very similar for the duration of the simulation.

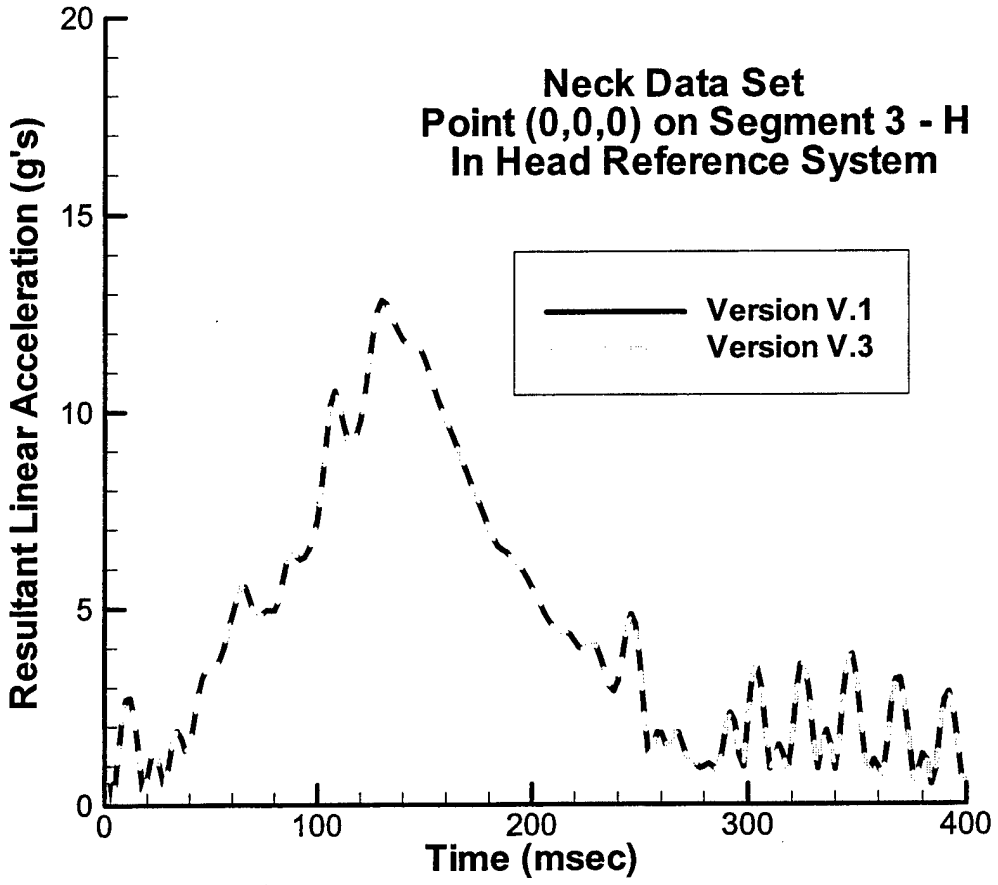

Figure 13 Neck Data Set Resultant Acceleration vs Time

3.1.11 S10INT Data Set

Two bodies totaling 34 segments and 33 joints are modeled. The vehicle models an S-10 pickup truck undergoing three rolls. The truck roof is given a prescribed motion to model its

deformation. Another prescribed motion models a window that breaks during the course of the simulation. The force environment consists of 39 contact planes and one harness for each of the occupants. This input deck was provided by the Air Force. No .AOU file listing is provided in the Appendices because of the length of the .AOU file. The simulation was run for 500 steps for a total simulation time of 1 second. Because of the complexity of this data set, the resultant acceleration of the centers of the head and upper torso of the first body and the upper torso of the second body were used as the comparison parameters. As can be seen from Figures 14, 15 and 16, overall the results for both versions of the code are very similar but differences in the results between the two versions do exist. These differences will be discussed in Section 3.4.

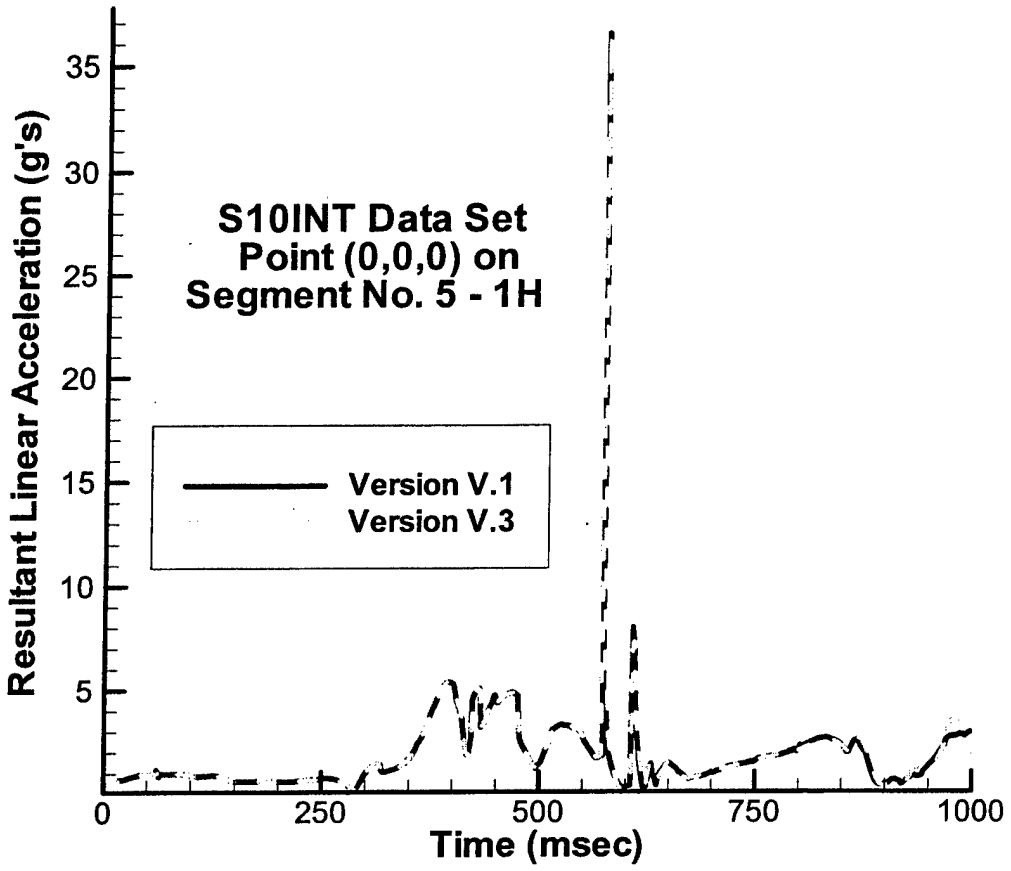

Figure 14 S10INT Data Set Head Resultant Acceleration vs Time - First Body

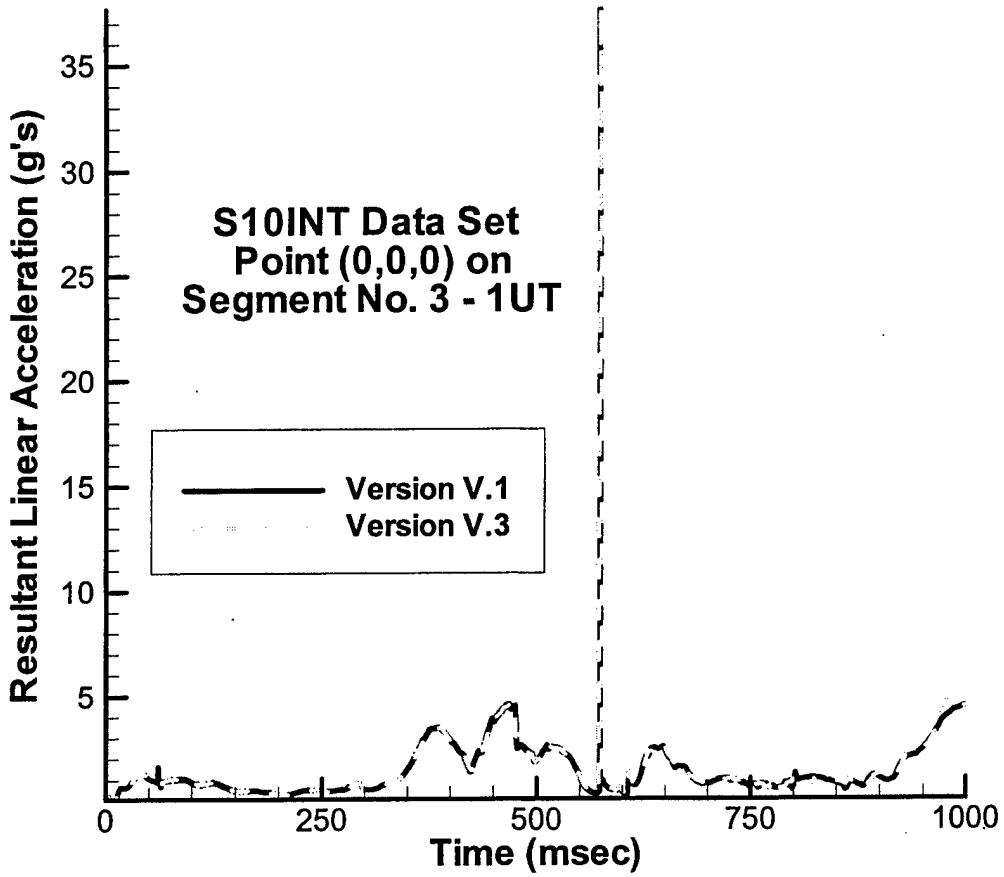

Figure 15 S10INT Data Set Upper Torso Resultant Acceleration vs Time - First Body

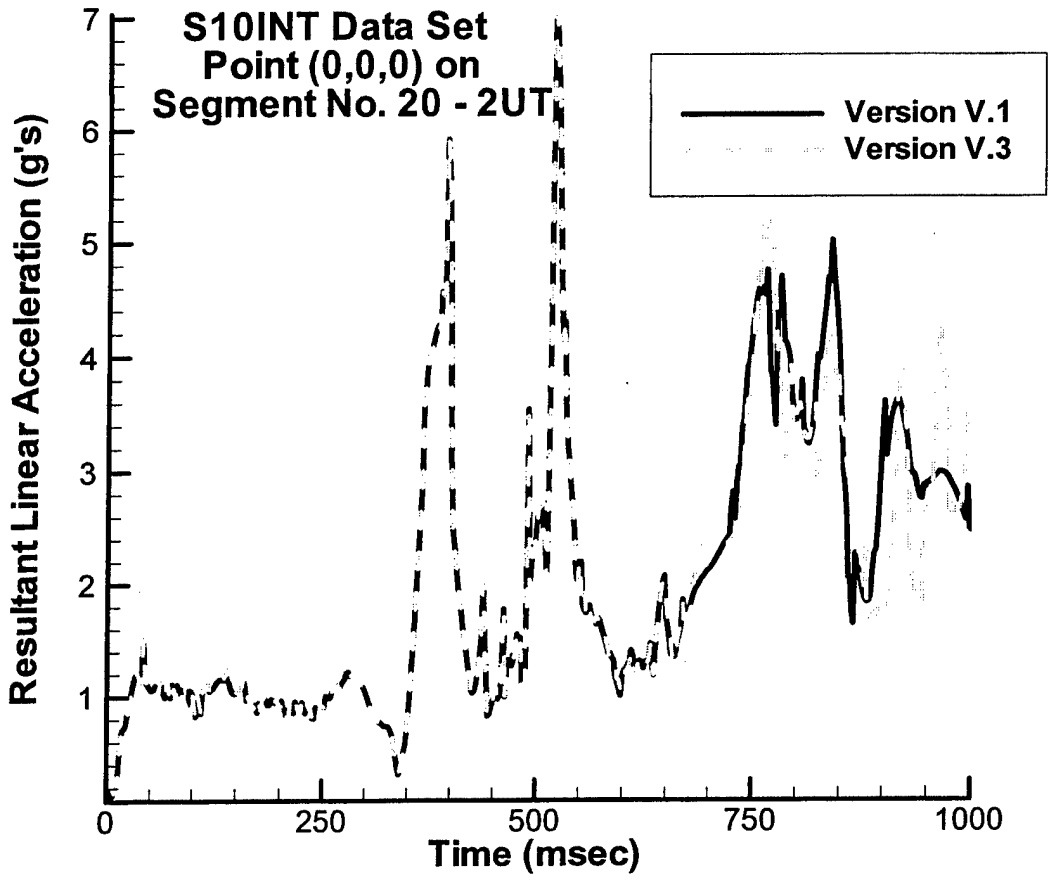

Figure 16 S10INT Data Set Upper Torso Resultant Acceleration vs Time - Second Body

3.1.12 S10INT_2 Data Set

Essentially the same as the S10INT input deck, except some of the contact plane force functions are different. This input deck was provided by the Air Force. No .AOU file listing is provided in the Appendices because of the length of the .AOU file. The simulation was run for 600 steps for a total simulation time of 1.2 seconds. The resultant acceleration of the head of the first body and the upper torso of the second body were used as the comparison parameters. As can be seen from Figures 17 and 18, the overall responses of both versions of the code are similar, but there exist significant differences between the two versions, especially after about 600 milliseconds into the simulation. These differences will be discussed in more detail in Section 3.4.

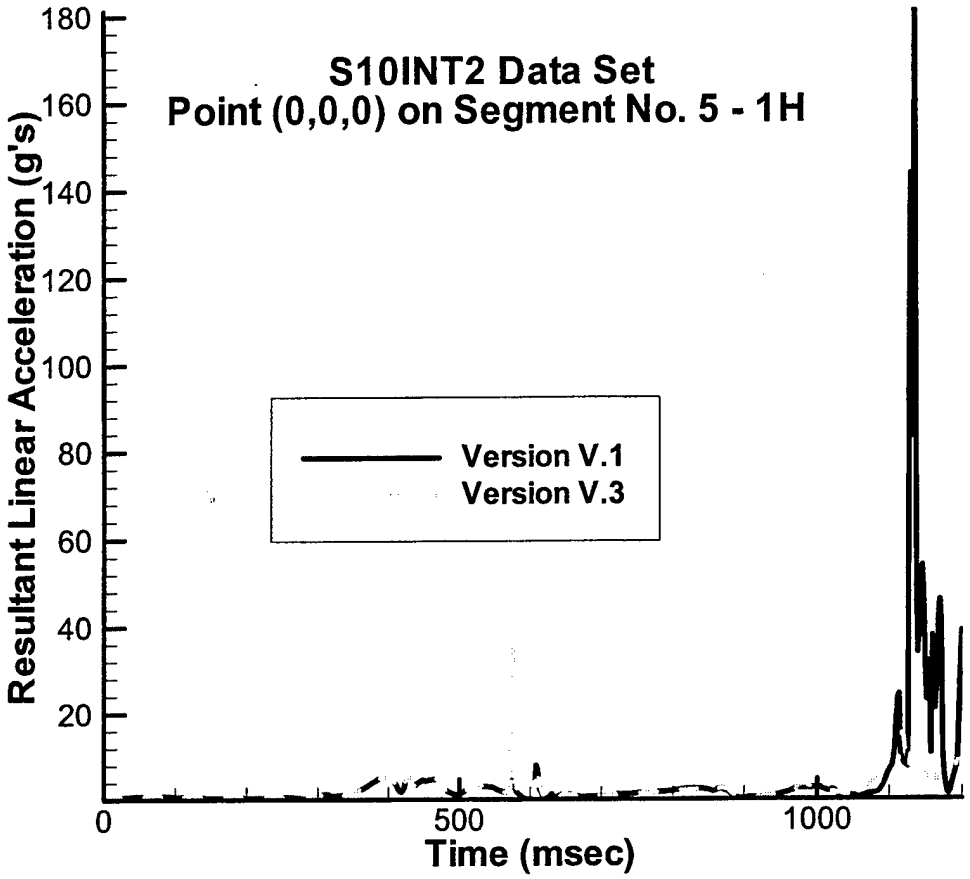

Figure 17 S10INT2 Data Set Head Resultant Acceleration vs Time - First Body

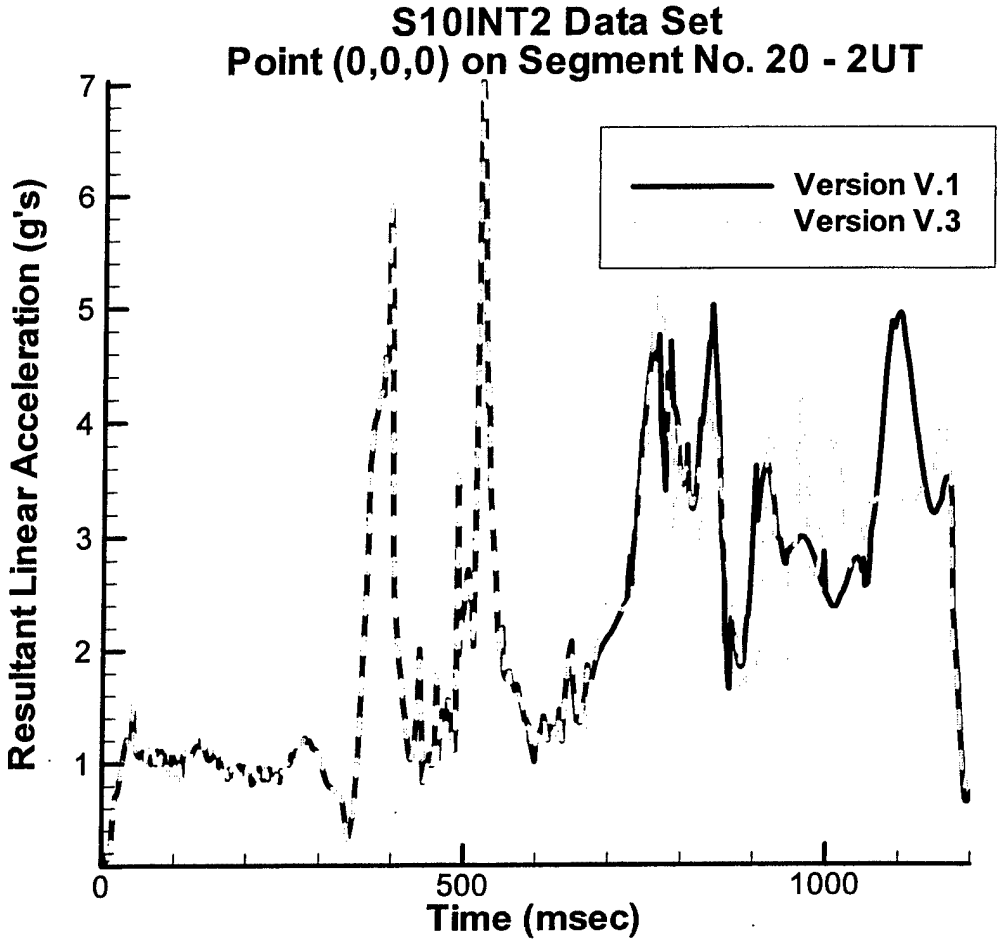

Figure 18 S10INT2 Data Set Upper Torson Resultant Acceleration vs Time - Second Body

3.1.13 Water Data Set

A single body composed of 17 segments and 16 joints representing a standing Hybrid III dummy is modeled. There is no vehicle motion. Water forces are applied to the dummy, with 1 regular wave and 1 personal flotation device. The body has no initial linear or angular velocities. This input deck was provided by the Air Force. A partial .AOU file listing is provided in the Appendices. The simulation was run for 40 steps for a total simulation time of 640 milliseconds. The resultant acceleration of the head was used as the comparison parameter. As can be seen from Figure 19, the results for both versions of the code are essentially identical up to approximately 420 milliseconds. After this time, there is a noticeable difference in magnitude between the two versions of the code.

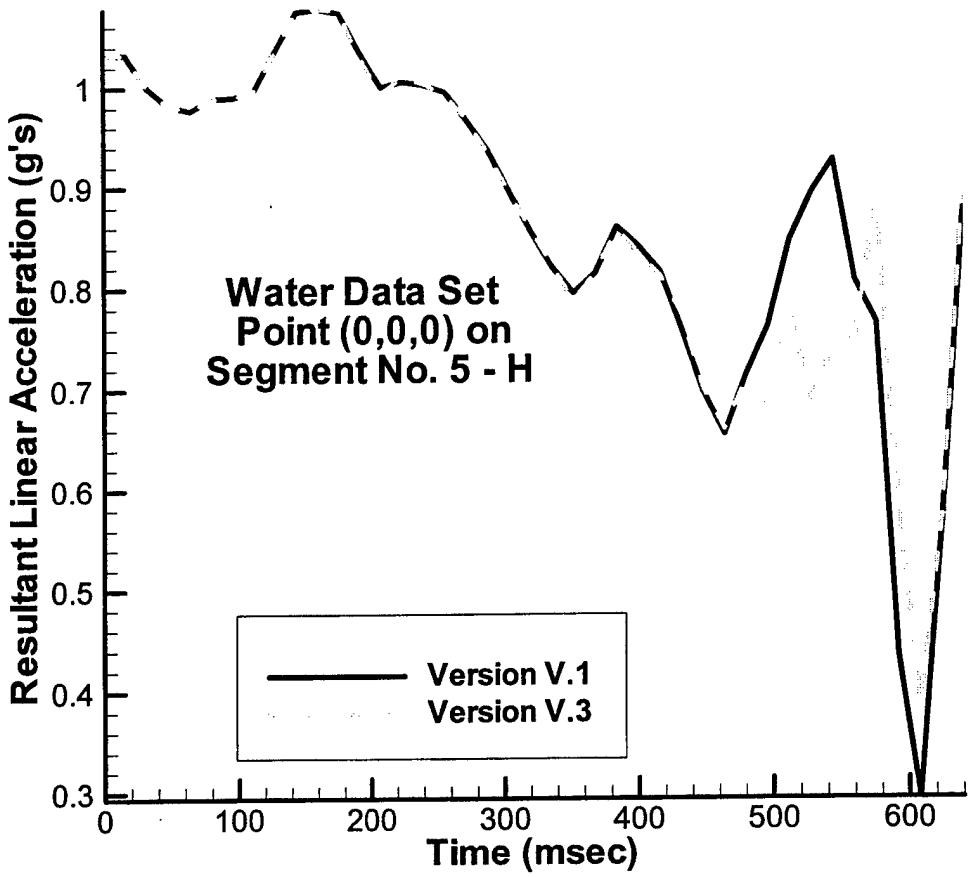

Figure 19 Water Data Set Resultant Acceleration vs Time

3.2 SGI Simulations

Several simulations using Version V.3 were run on a Silicon Graphics Indigo 2 workstation. The code developed on the PC compiler required no modifications to compile on the SGI. Simulation results were almost identical, with no differences exceeding fractions of a percent for any variable examined. It should be noted that only several of the many simulations run on the PC were actually run on the SGI. Therefore, any differences arising between machines because of numerical round-off errors were not thoroughly examined. It is recommended that additional simulations be performed on the SGI to better quantify the range of possible differences between the PC and the SGI due to numerical round-off.

3.3 Other Simulations

A large number of basic simulations were performed on the PC to test out many of the coding modifications and new features, such as the use of a KIND for variables and windowing of data output. Because of their basic nature, they are not reported here. Also, the code was compiled on a PC running Windows NT Workstation 4.0. Several simulations were run on the NT operating system and compared to the same simulations run a PC running either the Windows 95 or Windows 2000 operating system. As expected, the results were identical. Many additional simulations were run in an attempt to ascertain why there were differences between Version V.1 and Version V.3 of the code, as explained above. The results of these simulations are discussed below in Section 3.4.

3.4 Discussion of Results

In general, Version V.1 and Version V.3 produced very similar, if not identical, results for most of the simulations examined. This is to be expected, since none of the underlying mathematical algorithms used in the ATB Model were altered. Any differences that arise should be due solely to numerical round-off errors because of differences in how values are handled internally. These differences exist because a great deal of the coding logic was altered to eliminate GO TO statements, arithmetic IF statements and similar constructs. This type of recoding results in intermediate values being pushed on to the internal stacks and registers in slightly different sequences than was done for the original code. This will inevitably lead to slight differences in some values in the least significant bits.

3.4.1 Numerical Stability

With the large number of operations performed by the ATB Model in some complex simulations, it is possible for these very small differences to accumulate and grow with time until they may possibly alter the overall simulation results. It is felt that this is the cause for most of the minor differences observed in the simulations discussed in Sections 3.1.1 through 3.1.13. Small differences such as these should not be too big a concern if the code and algorithms are

numerically stable. However, after much examination, it was discovered that the one of the algorithms used in the harness belt algorithms is not numerically stable. An effect of O(12) was found, in the right circumstance, to produce a difference of O(1) in one integration step. [This means that a difference on the order of 10^{-12} can produce a result that differs by an order of 10^{-1}]. It is not unreasonable to expect that differences of O(12) can develop between the two versions of the code with time. This problem is demonstrated by the HYBIII data set (Section 3.1.6, Figures 6 and 7). However, when the appropriate harness belt algorithm was circumvented by requiring infinite friction between the harness belt and the contact ellipsoid, both versions of the code produced identical results, as can be seen in Figures 8 and 9.

Part of the differences observed in the S10INT andS10INT_2 data sets between the two versions of the code, Sections 3.1.11 and 3.1.12, respectively, may be due to the above mentioned problem. However, when the harness belt algorithm in question was circumvented, differences still remained. These differences were traced back to a different algorithm that contained logic that was similar to the harness belt algorithm and correspondingly proved to be numerically unstable. Given that there are at least two instances of numerically unstable algorithms contained within the code, and most likely others, it is impossible to expect different versions of the code to always produce identical results, even on the same machine. This problem is only compounded when one compares different compilers, different machines and different operating systems. This problem is further complicated by the use of different levels of optimization during compilation, which only tends to exacerbate the problem. The recommended solution is to make a careful and concerted effort to track down the numerically unstable algorithms contained within the code and to modify these algorithms with numerically stable coding techniques. The two equivalent versions of the code that now exist make an ideal test bed to perform this work. Once the known numerical stability problems are resolved, the effect of different levels of compiler optimization on simulation results can be examined.

3.4.2 Other Coding Problems

In the process of running the test simulations, several coding errors that are present in Version V.1 were encountered and corrected. The array NFBPR, which is now used by Subroutines INPUT_HCARDS and OUTPUT_HCARDS, had an incorrect counter. This is now corrected. In addition, the dimensions were wrong for the arrays IREFPL and IREFEL in Subroutine CHKREF. This error has also been corrected.

3.4.3 Current Limitations on Code Performance

Fortran 90, like many newer programming languages, is an object-oriented programming language that is conceptually "array object" oriented, whereas Fortran 77 is more "array element sequence" or "storage mapping" oriented. In Fortran 90, an array is considered an object in and of itself, as well as a sequence of related but separate elements [5, page 513]. Because of this philosophy, the programmer does not know how objects are stored in memory. It cannot be assumed that two arrays in a named common block are contiguous in memory as can be done in Fortran 77. Though it is beyond the scope of this report, a similar reasoning applies to passing of arguments between subroutines. Therefore, the common practice of passing arrays between ATB Model subroutines by passing only the first element of an array is discouraged in Fortran 90. This practice is discouraged because it goes against the spirit of object-oriented code by making an implicit assumption of how the arrays within each subroutine align in memory.

However, since Fortran 90 is required to accept all standard Fortran 77 code, this method of passing data between subroutines must be handled properly by a Fortran 90 compiler. To do so requires that the compiler include internal overhead in the compiled code to properly align the arrays. This additional overhead defeats the efficiency of the Fortran 90 compiler, and may slow down the code to the point that it may perform slower than the Fortran 77 code. This is unfortunate because several studies have shown the newer Fortran 90 compilers can create very efficient, compact and fast machine code for properly coded programs. The current code, even in the Fortran 90 version of the code, continues to use this outdated form of argument passing between subroutines. It is recommended that to increase the speed of Version V.3 of the code, arguments be passed as objects, rather than as array elements. The most efficient way to do this would be to further revise the code by forming structures composed of array objects that could then be passed as objects between subroutines. The use of structures would not only speed up the code but also make it more readable and easier to modify.

4. RECOMMENDATIONS

To maintain the usefulness of the coding modifications made under this effort, it is recommended that future software contracts include the following requirements.

4.1 Coding

a.) All new or modified subroutines should utilize the USE ONLY version of the USE statement. This will eliminate the inadvertent redefinition of a global variable in the module referenced by the USE statement if there is a local variable with the same name as a variable in the referenced module. Version V.3 of the code supplied under this contract has had all local variables with the same name as a global variable (a variable in any of the modules) renamed to clearly designate them as local variables. Future modifications of the code should continue this practice to clearly designate those variables that are local and to avoid confusion if a USE statement is inadvertently used instead of a USE ONLY statement. Furthermore, the use of the USE ONLY statement makes it easier to understand which global variables are used by a specific subroutine. Using the USE statement alone makes all the non-private variables in a module available to the subroutine, making it much harder to discern which of the global variables are used and/or altered by a particular subroutine.

b.) All new or modified subroutines should make use of the INTENT statements to clearly designate how each of the passed variables is to be treated. The use of these statements both makes it clearer to the programmer/reader of the code how the passed variables are to function within a subroutine, and forces the compiler to check that the variables are properly used. If the passed arguments are not used according their intent; i.e. only as input, only as output, or as input that can be modified for output, the compiler will flag this with a compile time error message.

c.) All new or modified subroutines should utilize the IMPLICIT NONE statement or flag in the compiler, i.e. all variables in a subprogram should be explicitly typed. This is in keeping with modern programming practice to explicitly list all variables used within a subprogram, eliminating inadvertent changing of variable types, and will catch any misspelled variable names during compilation, rather than defining the misspelled variable as a new variable with an initial value of 0 or blank.

d.) All new additions to the program should utilize modules rather than common blocks if the variables are not passed as arguments. As previously explained, the use of modules enables the compiler to enforce type and size consistency across all subroutines for any variables that are referenced within a specific module.

e.) All future modifications to the code should strive to follow modern coding conventions, such as limiting the use of GO TO statements, utilizing the CASE construct and IF/THEN blocks.

f.) The use of defining the KIND for each variable, both globally and locally should be continued, as well as defining the KIND of each constant. This will avoid the problem of Fortran 90/95 using incorrect values when initializing variables. As explained above, if the variable A has been explicitly typed as double precision, but the expression A = 1.0 is in the code, the rules of Fortran 90 state that the single precision value for 1.0 will be used instead of the double precision value. The proper expression is A = 1.0D0, or if double precision is defined by the KIND parameter I_HIGH, A = 1.0_I_HIGH.

g.) The use of structures within the code should be continued and expanded. The structure JOINT should be fully implemented and structures should be created for the force functions and the various contact options that are available in the program.

h.) The use of tabs within the code should be avoided, since tabs are not Fortran 90/95 standard.

i.) To maintain a visually consistent appearance of the code, operational code should be in uppercase whereas all comments should continue to be in lowercase with standard capitalization and grammar.

4.2 Use of Multiple Files

It is suggested that the code be maintained as individual subroutines within a specific subdirectory, rather than as a few very large files. Maintaining subroutines as individual files is supported by newer developer environments, such as the Microsoft Developer Studio™ [12,13]. The use of individual files permits faster editing times, quicker compiles, and the use of incremental linking, which as a whole result in a much shorter build time for the executable code. These developer environments permit searches across all the files within a specific directory, as well as find and replace options across files.

The use of individual files allows the newer developer environments to customize the compiler options by subroutines, such as setting debugging flags only where necessary during code debugging. Also, since source browsers are also provided in the newer developer environments, a programmer can easily check each instance where a variable is referenced, defined or used throughout the compiled code, with line numbers provided for each individual file/subroutine where such a reference is made.

Furthermore, the use of individual files makes it easier to build standard libraries of the code with only those subroutines needing special modifications for a specific machine, operating system or specialized version of the code having unique versions. Using the newer developer

environments, the programmer can develop make files that use the standard files/subroutines, linked with the necessary special files and the appropriate compiler-linker options to create an executable file targeted for a specific machine.

Finally, if the code is maintained as subroutines in individual files, minor coding modifications that involve only a few subroutines can be provided to users of the program by sending only the affected subroutines to individuals who have made customized modifications to the code. This avoids the annoyance of having to do search and compares between the previous version of the code and the new version of the code across files that have thousands of lines of code. It also makes it easier to track coding changes either manually or by the use of commercially available source code development tracking programs.

5. CONCLUSIONS

Though differences in results remain between Version V.3 of the ATB Model made under this effort and the current Version V.1, it is strongly recommended that the use of the Version V.1 code cease and be replaced with Version V.3. The majority of the differences between the two versions of the code seem to be due to numerical round-off errors that arise during a simulation because of numerically unstable algorithms, the most problematic being attributable to the friction forces for the harness belts and the contact planes.

The many advantages of Version V.3, a Fortran 90/95 version of the code, as explained in great detail throughout this report, make it the clear choice for the ATB Model coding for future versions of the model. Though a very large number of changes were made to the code, they are transparent to the general user. They are, however, far from transparent to anyone who needs to work with the code. The revised code is much easier to work with, much easier to follow and understand, more modular in nature, much less prone to errors and much easier to modify. Therefore, Version V.3 of the code should become the next standard version of the ATB Model and the recommendations given in Section 4 should be made into required guidelines to be followed for all future coding modifications to the ATB Model.

6. REFERENCES

1. *Validation of the Crash Victim Simulator: Volume 1 – Engineering Manual*, John T. Fleck and Frank E. Butler, CALSPAN Report No. ZS-5881-V-1, December 1981.

2. "Predictive Simulation of Restrained Occupant Dynamics in Vehicle Rollovers," J. Smith, A. Rizer and L. Obergefell, SAE 930887, 1993.

3. "Prediction of Whole-Body Response to Impact Forces in Flight Environments," I. Kaleps, *AGARD Conf. Proc. 253 – Models and Analogues for the Evaluation of Human Biodynamic Response, Performance and Protection*, Jan. 1979.

4. "The Use of the ATB/DYNAMAN in Injury Biomechanics," T. Khatua, L. Cheng, R. Fijan and R. Schmidt-Hargrave, *Proceedings of the 1995 ATB Model Users' Colloquium*, June 1995, Dayton, Ohio, 1995.

5 *Fortran 90 Handbook – Complete ANSI/ISO Reference*, J.C. Adams, W.S. Brainerd, J.T. Martin, B.T. Smith and J.L. Wagener, McGraw-Hill, New York, 1992.

6. *Programmer's Guide to Fortran 90*, W.S. Brainerd, C.H. Goldberg and J.C. Adams, McGraw-Hill, New York, 1990.

7. *Fortran 95 Handbook – Complete ISO/ANSI Reference*, J.C. Adams, W.S. Brainerd, J.T. Martin, B.T. Smith and J.L. Wagener, The MIT Press, Cambridge, MA, 1997.

8. *Fortran 95*, M. Counihan, UCL Press Limited, London, England, 1996.

9. "Parameterization of the Dynaman/ATB IV.2 Source Code", T.R. Gardner, GESAC final report, October 1991.

10. "Articulated Total Body Model Enhancements, Volume 2: User's Guide," L.A. Obergefell, T.R. Gardner, I. Kaleps and J.T. Fleck, AAMRL-TR-88-043, January 1988.

11. "Articulated Total Body Model Version V User's Manual," H. Cheng, A.L. Rizer and L.A. Obergefell, Report No. AFRL-HE-WP-TR-1998-0015, February 1998.

12. *Digital Visual Fortran Programmer's Guide*, Digital Press, M. Etzel and K. Dickinson, Butterworth-Heinemann, Woburn, MA, 1999.

13. *DIGITAL Fortran Language Reference Manual*, Digital Equipment Corporation, Maynard MA, 1997.

THIS PAGE LEFT BLANK INTENTIONALLY

Appendix A - Module Source Code

A.1 MODULE_STANDARD Source Code

```
      MODULE MODULE_STANDARD
!C
!C                                                    Rev. V.3 7/10/2004
!C
!C    This module contains all the COMMON BLOCKs and PARAMETERS
!C     found in BLOCKDATA in version ATBV_1.
!C
      IMPLICIT NONE
!C
!C*****************************************************************************
!C    Define the KIND parameters for use throughout the program.
!C*****************************************************************************
!C
      INTEGER       ICHAR_STD, INTEGER_STD,  IREAL_HIGH, IREAL_STD,
     &              LOGICAL_STD
      PARAMETER   ( ICHAR_STD = 1, INTEGER_STD = 4, IREAL_HIGH = 8,
     &              IREAL_STD = 4, LOGICAL_STD = 4 )
!C
!C*****************************************************************************
!C    Define the numerical constants for the program.
!C*****************************************************************************
!C
      INTEGER ( KIND = INTEGER_STD )  I_0, I_1, I_2, I_3, I_4, I_5,
     &                                I_6, I_7, I_8, I_9, I_10, I_11,
     &                                I_12, I_13, I_14, I_15, I_16,
     &                                I_18, I_20, I_45, I_50, I_100
      PARAMETER   ( I_0     =   0_INTEGER_STD,
     &              I_1     =   1_INTEGER_STD,
     &              I_2     =   2_INTEGER_STD,
     &              I_3     =   3_INTEGER_STD,
     &              I_4     =   4_INTEGER_STD,
     &              I_5     =   5_INTEGER_STD,
     &              I_6     =   6_INTEGER_STD,
     &              I_7     =   7_INTEGER_STD,
     &              I_8     =   8_INTEGER_STD,
     &              I_9     =   9_INTEGER_STD,
     &              I_10    =  10_INTEGER_STD,
     &              I_11    =  11_INTEGER_STD,
     &              I_12    =  12_INTEGER_STD,
     &              I_13    =  13_INTEGER_STD,
     &              I_14    =  14_INTEGER_STD,
     &              I_15    =  15_INTEGER_STD,
     &              I_16    =  16_INTEGER_STD,
     &              I_18    =  18_INTEGER_STD,
     &              I_20    =  20_INTEGER_STD,
     &              I_45    =  45_INTEGER_STD,
     &              I_50    =  50_INTEGER_STD,
     &              I_100   = 100_INTEGER_STD  )

!C
      REAL    ( KIND = IREAL_STD )  R_0, R_HALF, R_100, R_1000
      PARAMETER ( R_0          =    0.0E0_IREAL_STD,
     &            R_HALF       =    0.5E0_IREAL_STD,
     &            R_100        =  100.0E0_IREAL_STD,
     &            R_1000       = 1000.0E0_IREAL_STD )
!C
      REAL    ( KIND = IREAL_HIGH )   D_0, D_THIRD, D_HALF, D_1,
     &                                D_2, D_3, D_4, D_5, D_6, D_8,
     &                                D_9, D_10, D_12,
     &                                D_100, D_180, D_1000
      PARAMETER ( D_0         =    0.0E0_IREAL_HIGH,
     &            D_THIRD     =    1.0E0_IREAL_HIGH / 3.0E0_IREAL_HIGH,
     &            D_HALF      =    0.5E0_IREAL_HIGH,
     &            D_1         =    1.0E0_IREAL_HIGH,
     &            D_2         =    2.0E0_IREAL_HIGH,
     &            D_3         =    3.0E0_IREAL_HIGH,
     &            D_4         =    4.0E0_IREAL_HIGH,
     &            D_5         =    5.0E0_IREAL_HIGH,
     &            D_6         =    6.0E0_IREAL_HIGH,
```

```
     &            D_8         =    8.0E0_IREAL_HIGH,
     &            D_9         =    9.0E0_IREAL_HIGH,
     &            D_10        =   10.0E0_IREAL_HIGH,
     &            D_12        =   12.0E0_IREAL_HIGH,
     &            D_100       =  100.0E0_IREAL_HIGH,
     &            D_180       =  180.0E0_IREAL_HIGH,
     &            D_1000      = 1000.0E0_IREAL_HIGH  )
!C
!C***************************************************************************
!C    Define the program name, version number and version date.
!C***************************************************************************
!C
      CHARACTER ( LEN = 4, KIND = ICHAR_STD ) PROGRAM_NAME
      PARAMETER ( PROGRAM_NAME = ICHAR_STD_'ATB' )
      CHARACTER ( LEN = 14, KIND = ICHAR_STD ) VERSION_NUMBER
      PARAMETER ( VERSION_NUMBER = ICHAR_STD_'Version 5.3.1 ' )
      CHARACTER ( LEN = 17, KIND = ICHAR_STD ) VERSION_DATE
      PARAMETER ( VERSION_DATE = ICHAR_STD_'July 10, 2004' )
!C
!C***************************************************************************
!C    Define the global constants used in ATB Model version V_3.
!C***************************************************************************
!C
!C    MAXSEG  = Maximum no. of segments
!C    MAXJNT  = Maximum no. of joints
!C    MAXELP  = Maximum no. of ellipsoids
!C    MAXXELP = maximun no. of additional ellipsoids beyond the default
!C              of one contact ellipsoid per segment
!C
      INTEGER  ( KIND = INTEGER_STD ) MAXSEG, MAXJNT, MAXELP, MAXXELP
      PARAMETER ( MAXXELP = I_20 )
      PARAMETER ( MAXSEG = 60_INTEGER_STD, MAXJNT = MAXSEG,
     &            MAXELP = MAXSEG + MAXXELP )
!C
!C    MAXPLN = Maximum no. of contact planes
!C    MAXPSF = maximum no. of plane/segment contacts
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXPLN, MAXPSF
      PARAMETER ( MAXPLN = I_50, MAXPSF = 200_INTEGER_STD )
!C
!C    MAX_NUM_BELTS = Maximum no. of simple belts
!C
      INTEGER  ( KIND = INTEGER_STD ) MAX_NUM_BELTS
      PARAMETER ( MAX_NUM_BELTS = I_20 )
!C
!C    MAX_NUM_DAMPERS = Maximum no. of spring dampers
!C
      INTEGER  ( KIND = INTEGER_STD )  MAX_NUM_DAMPERS
      PARAMETER ( MAX_NUM_DAMPERS = I_20 )
!C
!C    MAXSSF = maximum no. of segment/segment contacts
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXSSF
      PARAMETER ( MAXSSF = 150_INTEGER_STD )
!C
!C    MAXHRN  = maximum no. of harness belt systems
!C    MAXHBLT = maximum no. of harness belts from all of the harnesses combined
!C    MAXHPH  = maximum no. of belt points per harness belt system
!C    MAXHPT  = maxinum no. of belt points from all of the harnesses combined
!C     MULHRN  = multiplier used for tie point
!C
      INTEGER  ( KIND = INTEGER_STD ) MAXHRN, MAXHBLT, MAXHPH, MAXHPT
       INTEGER  ( KIND = INTEGER_STD ) MULHRN
      PARAMETER ( MAXHRN = I_5,  MAXHBLT = I_20,
     &            MAXHPH = I_50, MAXHPT = I_100, MULHRN = I_100 )
!C
!C    MAX_FOR_TORQ = maximum number of force/torque functions that are
!C     specified by the D.9 cards.
!C
      INTEGER  ( KIND = INTEGER_STD ) MAX_FOR_TORQ
      PARAMETER ( MAX_FOR_TORQ = I_5 )
```

```
!C
!C    MAXREF = maximun no. of reference segments, i.e. segments for which full
!C             6 degree of freedom motion is computed
!C    MAXEQN = maximum no. of vector state variables
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXREF, MAXEQN
      PARAMETER ( MAXREF = I_20 )
      PARAMETER ( MAXEQN = I_2 * ( MAXSEG + MAXREF ) )
!C
!C    MAX_FUNC = maximum no. of functions
!C    MAXNTB   = maximum no. of elements in the NTAB array.  The scale factor
!C               of 75 * MAX_FUNC is based on previous experience and may need
!C               to be increased.
!C    MAXTAB   = maximum no. of elements in the TAB array. The scale factor
!C               of 270 * MAX_FUNC is based on previous experience and may need
!C               to be increased.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAX_FUNC, MAXNTB, MAXTAB
      PARAMETER ( MAX_FUNC = 98_INTEGER_STD,
     &            MAXNTB   = 75_INTEGER_STD * MAX_FUNC,
     &            MAXTAB   = 270_INTEGER_STD * MAX_FUNC )
!C
!C    MHEDNG = number of elements needed by arrays NOPL, MOPL, and M1PL
!C             during postprocessing
!C             In the following expression, The 1st '20' refers to maximum
!C             no. of simple belt and harness belt endpoint strains and
!C             forces.  The 2nd '20' refers to maximum amount of
!C             airbag/contact force information.
!C
      INTEGER  ( KIND = INTEGER_STD )  MHEDNG
      PARAMETER ( MHEDNG = MAXPSF + I_20 + MAXSSF + I_20 )
!C
!C    MAXCST = maximum no. of constraints.
!C
      INTEGER  ( KIND = INTEGER_STD )   MAXCST
      PARAMETER ( MAXCST = I_12 )
!C
!C    MAXFLX = maximum no. of flexible elements.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXFLX
      PARAMETER ( MAXFLX = I_8 )
!C
!C    MAXRHS = maximum no. of equations which can be solved by the program.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXRHS
      PARAMETER ( MAXRHS = I_2 * MAXJNT + MAXFLX + MAXCST )
!C
!C    MAXCMX = maximum size of the solution matrix, C.
!C             Set the maximum size of the solution matrix, C, to approximately
!C             20.5% of the fully populated matrix that corresponds to the
!C             full length of the known vector, RHS.  The 20.5% populated size
!C             for the C matrix was selected from previous experience with
!C             how populated C was for various ATB simulations.  Approximate
!C             it generously by 1/4.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXCMX
      PARAMETER ( MAXCMX = ( MAXRHS**I_2 / I_4 ) )
!C
!C    MXMOD = maximum no. of mode shapes used for deformable segments.
!C    MXNOD = maximum no. of nodes in a deformable segment.
!C    MAXDEF = maximum no. of deformable segments
!C
      INTEGER  ( KIND = INTEGER_STD )  MXNOD, MXMOD, MAXDEF
      PARAMETER ( MXNOD = 4000_INTEGER_STD, MXMOD = I_4,
     &            MAXDEF = I_1 )
!C
!C    MXHIC = maximum no. of time points stored for HIC computation
!C
      INTEGER  ( KIND = INTEGER_STD )  MXHIC
      PARAMETER ( MXHIC = 6000_INTEGER_STD )
!C
```

```
!C    MAXBAG = maximum no. of airbags.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXBAG
      PARAMETER ( MAXBAG = I_5 )
!C
!C    MAXBDP =  ??????
!C    MAXBDT =  ??????
!C    MAXBSH =  ??????
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXBDP, MAXBDT, MAXBSH
      PARAMETER ( MAXBDP = 400_INTEGER_STD, MAXBDT = I_20,
     &            MAXBSH = I_20 )
!C
!C    MAXVEH = maximum no. of vehicles.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXVEH
      PARAMETER ( MAXVEH = I_6 )
!C
!C    MAXVT2 = maximum number of time points for vehicle input option 2.
!C    MAXVT3 = maximum number of time points for vehicle input option 3.
!C    MAXVT4 = maximum number of time points for vehicle input option 4.
!C    MAXVDT = maximum number of time (data) points for vehicle motion.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAXVT2, MAXVT3, MAXVT4, MAXVDT
      PARAMETER ( MAXVT2 = 99_INTEGER_STD, MAXVT3 = 5001_INTEGER_STD,
     &            MAXVT4 = 5001_INTEGER_STD, MAXVDT = 5001_INTEGER_STD )
!C
!C    MAX_NUM_TTH = maximum number of tabular time history files.
!C
      INTEGER  ( KIND = INTEGER_STD)  MAX_NUM_TTH
      PARAMETER ( MAX_NUM_TTH = 200_INTEGER_STD )
!C
!C    MAX_HCARD_TTH = maximum number of tabular time histories for
!C     H.1 through H.9 and H.11 cards.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAX_HCARD_TTH
      PARAMETER ( MAX_HCARD_TTH = I_20 )
!C
!C    MAX_TOTAL_BODY = maximum number of bodies for the total body
!C     properties specified by the H.10 cards.
!C
      INTEGER  ( KIND = INTEGER_STD )  MAX_TOTAL_BODY
      PARAMETER ( MAX_TOTAL_BODY = I_5 )
!C
!C    MAX_LN_PPAGE = maximum number of lines of data per page of
!C     the tabular time history, when it is outputted in "paged"
!C     format, i.e. it has a heading for each page.
!C
      INTEGER  ( KIND = INTEGER_STD ) MAX_LN_PPAGE
      PARAMETER ( MAX_LN_PPAGE = I_45 )
!C
!C    Offset value after which the tabular time history logical units
!C     start.
!C
      INTEGER  ( KIND = INTEGER_STD)  NUM_TTH_OFFSET
      PARAMETER ( NUM_TTH_OFFSET = I_20 )
!C
!C    Define the logical unit numbers where:
!C       LUAIN      is the standard input unit;
!C       LUAOU      is the standard output unit;
!C       LUDEBUG    is the debug output unit;
!C       LUTP8      is the unformatted file used for postprocessing;
!C       LUVIEW     is the VIEW output unit;
!C       LUFLEX     is for the flexible segment input unit;
!C       LU_MEM     is the memory (run parameters) input/output unit;
!C       LUFACET    is for the faceted surface input unit;
!C       LUTERM_IN  is for terminal/console input;
!C       LUTERM_OUT is for terminal/console output.
!C
      INTEGER  ( KIND = INTEGER_STD )  LUAIN, LUAOU, LUTP8, LUDEBUG,
     &                                 LUVIEW, LUFLEX, LUFACET, LULIN,
```

```
      &                                    LUTERM_IN, LUTERM_OUT, LU_MEM
       PARAMETER ( LUTERM_OUT =  I_0,
      &            LUVIEW     =  I_1,
      &            LU_MEM     =  I_2,
      &            LULIN      =  I_3,
      &            LUAIN      =  I_4,
      &            LUTERM_IN  =  I_5,
      &            LUAOU      =  I_6,
      &            LUDEBUG    =  I_7,
      &            LUTP8      =  I_8,
      &            LUFLEX     =  I_11,
      &            LUFACET    =  I_12 )
!C
!C    Define logical constants for program.
!C
      LOGICAL ( KIND = LOGICAL_STD )  FALSE, TRUE
      PARAMETER ( FALSE = .FALSE._LOGICAL_STD,
     &            TRUE  = .TRUE._LOGICAL_STD )
!C
!C    Define the logical control parameters for the type
!C     of input to be supplied:
!C       LIN_FLAG - true for list-directed input, i.e. .LIN;
!C                - false for explicitly formatted input, i.e. .AIN
!C     and whether a .LIN file is to be created from a .AIN file:
!C       AIN_CONVERT - true to create a .LIN file
!C                   - false, no .LIN file will be created.
!C
      LOGICAL ( KIND = LOGICAL_STD )  LIN_FLAG, AIN_CONVERT
!C
!C
!C*****************************************************************************
!C    Define the Structures
!C*****************************************************************************
!C-----------------------------------------------------------------------------
!C    Define the structure type for tabular time histories.
!C          BEGIN_LUNUM - beginning logical unit number for a particular type
!C                        of tabular time history output;
!C          END_LUNUM   - ending logical unit number for a particular type
!C                        of tabular time history output;
!C          PRINT       - logical flag; if true, the tabular time history
!C                        is to be outputted, if false, the tabular time
!C                        will not be outputted.
!C
      TYPE  TTH_DESCRIP
       INTEGER  ( KIND = INTEGER_STD )  BEGIN_LUNUM
       INTEGER  ( KIND = INTEGER_STD )  END_LUNUM
       LOGICAL  ( KIND = LOGICAL_STD )  PRINT
      END TYPE  TTH_DESCRIP
!C
!C    Define the tabular time history descriptors - the names are self-explanatory.
!C
      TYPE ( TTH_DESCRIP )  ACUATOR_TTH
      TYPE ( TTH_DESCRIP )  AIRBAG_TTH
      TYPE ( TTH_DESCRIP )  ANG_ACCEL_TTH
      TYPE ( TTH_DESCRIP )  ANG_ROT_TTH
      TYPE ( TTH_DESCRIP )  ANG_VEL_TTH
      TYPE ( TTH_DESCRIP )  BELT_TTH
      TYPE ( TTH_DESCRIP )  CG_TTH
      TYPE ( TTH_DESCRIP )  HARNESS_TTH
      TYPE ( TTH_DESCRIP )  JOINT_FORCE_TTH
      TYPE ( TTH_DESCRIP )  JOINT_PARM_TTH
      TYPE ( TTH_DESCRIP )  LIN_ACCEL_TTH
      TYPE ( TTH_DESCRIP )  LIN_DISP_TTH
      TYPE ( TTH_DESCRIP )  LIN_VEL_TTH
      TYPE ( TTH_DESCRIP )  PLANE_SEG_TTH
      TYPE ( TTH_DESCRIP )  SEG_SEG_TTH
      TYPE ( TTH_DESCRIP )  SPRING_DAMP_TTH
      TYPE ( TTH_DESCRIP )  WATER_TTH
      TYPE ( TTH_DESCRIP )  WIND_TTH
!C
!C-----------------------------------------------------------------------------
```

```
!C    Define the structure and parameters for
!C     time-windowing the output.
!C
!C     MAX_NUM_OUT_TIMES - the maximun number of time windows permitted,
!C     NUM_OUT_TIMES     - the number of time windows for a particular run.
!C
      INTEGER   ( KIND = INTEGER_STD)  MAX_NUM_OUT_TIMES
      INTEGER   ( KIND = INTEGER_STD)  NUM_OUT_TIMES
      PARAMETER ( MAX_NUM_OUT_TIMES = I_3 )
!C
!C      For this structure:
!C            PRINT - a logical flag; if true, the time windowing option
!C                    applies to output from Subroutine PRINT, if false,
!C                    time windowing is ignored by Subroutine PRINT
!C            TTH   - a logical flag; if true, the time windowing option
!C                    applies to the tabular time history output, if false,
!C                    time windowing is ignored for the tabular time histories
!C            VIEW  - a logical flag; if true, the time windowing option
!C                    applies to output for VIEW output, if false, time
!C                    windowing is ignored for VIEW output
!C
!C            START - the time, in seconds, when the time window begins
!C                    for data to be outputted, this time is the simulation
!C                    time, i.e. relative to time = 0.0 when the integration
!C                    begins for a particular simulation
!C            END   - the time, in seconds, when the time window stops, this
!C                    time is the simulation time
!C
      TYPE OUTPUT_TIMES
       LOGICAL  ( KIND = LOGICAL_STD )  PRINT
       LOGICAL  ( KIND = LOGICAL_STD )  TTH
       LOGICAL  ( KIND = LOGICAL_STD )  VIEW
       REAL ( KIND = IREAL_HIGH )  START
       REAL ( KIND = IREAL_HIGH )  END
      END TYPE OUTPUT_TIMES
!C
!C    Define the time control structure.
!C
      TYPE ( OUTPUT_TIMES) OUT_TIMES(MAX_NUM_OUT_TIMES)
!C
!C-------------------------------------------------------------------------------
!C    Define the segment structure.
!C
      TYPE SEGMENT
       REAL      ( KIND = IREAL_HIGH )          ANG_ACCEL(3)    ! was WMEGD
       REAL      ( KIND = IREAL_HIGH )          ANG_ACL_CONV(3)! was SGTEST(,3,)
       REAL      ( KIND = IREAL_HIGH )          ANG_VEL(3)      ! was WMEG
       REAL      ( KIND = IREAL_HIGH )          ANG_VEL_CONV(3)! was SGTEST(,1,)
       REAL      ( KIND = IREAL_HIGH )          DIR_COS(3,3)    ! was D
       REAL      ( KIND = IREAL_HIGH )          DRC_PHI(3,3)    ! was DPMI
       REAL      ( KIND = IREAL_HIGH )          EXT_ANG_ACL(3)  ! was U2
       REAL      ( KIND = IREAL_HIGH )          EXT_LIN_ACL(3)  ! was U1
       REAL      ( KIND = IREAL_HIGH )          LIN_ACCEL(3)    ! was SEGLA
       REAL      ( KIND = IREAL_HIGH )          LIN_ACL_CONV(3)! was SGTEST(,4,)
       REAL      ( KIND = IREAL_HIGH )          LIN_DISP(3)     ! was SEGLP
       REAL      ( KIND = IREAL_HIGH )          LIN_VEL(3)      ! was SEGLV
       REAL      ( KIND = IREAL_HIGH )          LIN_VEL_CONV(3)! was SGTEST(,2,)
       CHARACTER ( LEN = 4, KIND = ICHAR_STD )  NAME            ! was SEG
       REAL      ( KIND = IREAL_HIGH )          PHI(3)          ! was PHI
       REAL      ( KIND = IREAL_HIGH )          RECIP_MASS      ! was RW
       REAL      ( KIND = IREAL_HIGH )          RECIP_PHI(3)    ! was RPHI
       LOGICAL   ( KIND = LOGICAL_STD )         ROT_PHI         ! was LPMI
       INTEGER   ( KIND = INTEGER_STD )         SINGULAR        ! was ISING
       REAL      ( KIND = IREAL_HIGH )          WEIGHT          ! was W
      END TYPE SEGMENT
!C
      TYPE ( SEGMENT ) SEG
      DIMENSION        SEG(MAXSEG)
!C
!C-------------------------------------------------------------------------------
!C    Define the joint structure.  Commented out items within the
```

```
!C     structure have not yet been implemented.
!C
       TYPE JOINT
        CHARACTER ( LEN = 4,
      &             KIND = ICHAR_STD )     JNT_NAME          ! was JOINT
        INTEGER   ( KIND = INTEGER_STD )    PROX_SEG          ! was JNT
        LOGICAL   ( KIND = LOGICAL_STD )    EULER             ! was EULER
        LOGICAL   ( KIND = LOGICAL_STD )    SLIP_FREE         ! was FREE
!c        INTEGER   ( KIND = INTEGER_STD )    DSTL_SEG          ! new J+1
        REAL      ( KIND = IREAL_HIGH )     PROX_LOC(3)       ! was SR(1-3,2*J-1)
        REAL      ( KIND = IREAL_HIGH )     DSTL_LOC(3)       ! was SR(1-3,2*J)
        REAL      ( KIND = IREAL_HIGH )     PROX_CNST         ! was SR(4,2*J-1)
        REAL      ( KIND = IREAL_HIGH )     DSTL_CNST         ! was SR(4,2*J)
        INTEGER   ( KIND = INTEGER_STD )    JTYPE             ! was IPIN
!c        INTEGER   ( KIND = INTEGER_STD )    IEULER            ! was IEULER
!c        INTEGER   ( KIND = INTEGER_STD )    ISLIP             ! was ISLIP
        REAL      ( KIND = IREAL_HIGH )     TENS_MAX          ! was CONST(1,)
        REAL      ( KIND = IREAL_HIGH )     COMP_MAX          ! was CONST(2,)
!c        REAL      ( KIND = IREAL_HIGH )     PROX_COORD(3,3)   ! was HT(3,3,2*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     DSTL_COORD(3,3)   ! was HT(3,3,2*J)
!c        INTEGER   ( KIND = INTEGER_STD )    PROX_NODE(3)      ! was NODJ(,1)
        REAL      ( KIND = IREAL_HIGH )     PROX_HA(3)         ! was HA(3,2*J-1)
        REAL      ( KIND = IREAL_HIGH )     DSTL_HA(3)         ! was HA(3,2*J)
        REAL      ( KIND = IREAL_HIGH )     PROX_HB(3)         ! was HB(3,2*J-1)
        REAL      ( KIND = IREAL_HIGH )     DSTL_HB(3)         ! was HB(3,2*J)
!c        INTEGER   ( KIND = INTEGER_STD )    DSTL_NODE(3)      ! was NODJ(,2)
!C
!c        REAL      ( KIND = IREAL_HIGH )     PROX_ROT(3)       ! was YPR1
!c        REAL      ( KIND = IREAL_HIGH )     DSTL_ROT(3)       ! was YPR2
!c        INTEGER   ( KIND = INTEGER_STD )    PROX_SEQ(3)       ! was IDYPR(1-3)
!c        INTEGER   ( KIND = INTEGER_STD )    DSTL_SEQ(3)       ! was IDYPR(4-6)
!c        REAL      ( KIND = IREAL_HIGH )     INIT_ROT_ANG(3)   ! was ANG(3)
       REAL      ( KIND = IREAL_HIGH )     CNTR_SYM(3)       ! was CONST(1-3,)
       REAL      ( KIND = IREAL_HIGH )     COS_NUTA          ! was CONST(4,)
       REAL      ( KIND = IREAL_HIGH )     SIN_NUTA          ! was CONST(5,)
!C
!c        REAL      ( KIND = IREAL_HIGH )     FLX_LIN_COEF      ! was SPRING(,1,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     FLX_QUA_COEF      ! was SPRING(,2,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     FLX_CUB_COEF      ! was SPRING(,3,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     FLX_ENRG_DIS      ! was SPRING(,4,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     FLX_JNT_STOP      ! was SPRING(,5,3*J-2)
!C
!c        REAL      ( KIND = IREAL_HIGH )     TRQ_LIN_COEF      ! was SPRING(,1,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     TRQ_QUA_COEF      ! was SPRING(,2,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     TRQ_CUB_COEF      ! was SPRING(,3,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     TRQ_ENRG_DIS      ! was SPRING(,4,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     TRQ_JNT_STOP      ! was SPRING(,5,3*J-1)
!C
!c        REAL      ( KIND = IREAL_HIGH )     SPN_LIN_COEF      ! was SPRING(,1,3*J)
!c        REAL      ( KIND = IREAL_HIGH )     SPN_QUA_COEF      ! was SPRING(,2,3*J)
!c        REAL      ( KIND = IREAL_HIGH )     SPN_CUB_COEF      ! was SPRING(,3,3*J)
!c        REAL      ( KIND = IREAL_HIGH )     SPN_ENRG_DIS      ! was SPRING(,4,3*J)
!c        REAL      ( KIND = IREAL_HIGH )     SPN_JNT_STOP      ! was SPRING(,5,3*J)
!C
!c        REAL      ( KIND = IREAL_HIGH )     PRE_VIS_COEF      ! was VISC(1,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     PRE_COU_FRIC      ! was VISC(2,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     PRE_REL_ANG_VEL   ! was VISC(3,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     PRE_MAX_TORQ      ! was VISC(4,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     PRE_MIN_TORQ      ! was VISC(5,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     PRE_MIN_ANG_VEL   ! was VISC(6,3*J-2)
!c        REAL      ( KIND = IREAL_HIGH )     PRE_COEF_RST      ! was VISC(7,3*J-2)
!C
!c        REAL      ( KIND = IREAL_HIGH )     NUT_VIS_COEF      ! was VISC(1,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     NUT_COU_FRIC      ! was VISC(2,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     NUT_REL_ANG_VEL   ! was VISC(3,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     NUT_MAX_TORQ      ! was VISC(4,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     NUT_MIN_TORQ      ! was VISC(5,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     NUT_MIN_ANG_VEL   ! was VISC(6,3*J-1)
!c        REAL      ( KIND = IREAL_HIGH )     NUT_COEF_RST      ! was VISC(7,3*J-1)
!C
!c        REAL      ( KIND = IREAL_HIGH )     SPN_VIS_COEF      ! was VISC(1,3*J)
```

```
!c       REAL       ( KIND = IREAL_HIGH )    SPN_COU_FRIC     ! was VISC(2,3*J)
!c       REAL       ( KIND = IREAL_HIGH )    SPN_REL_ANG_VEL  ! was VISC(3,3*J)
!c       REAL       ( KIND = IREAL_HIGH )    SPN_MAX_TORQ     ! was VISC(4,3*J)
!c       REAL       ( KIND = IREAL_HIGH )    SPN_MIN_TORQ     ! was VISC(5,3*J)
!c       REAL       ( KIND = IREAL_HIGH )    SPN_MIN_ANG_VEL  ! was VISC(6,3*J)
!c       REAL       ( KIND = IREAL_HIGH )    SPN_COEF_RST     ! was VISC(7,3*J)
!C
        REAL       ( KIND = IREAL_HIGH )    JFORCE(3)        ! was F
        REAL       ( KIND = IREAL_HIGH )    JTORQUE(3)       ! was TQ
      END TYPE JOINT
!C
      TYPE ( JOINT ) JNT
      DIMENSION      JNT(MAXJNT)
!C
!C-------------------------------------------------------------------------------
!C     Define the vehicle structure:
!C
!C       VNAME       = name of the vehicle
!C       VTITLE      = description of the vehicle
!C       IVFLG       = flag denoting whether the prescribed motion is
!C                     relative to the ground (0) or a segment (1 or 2)
!C       IVSEG       = vehicle segment number
!C       IVREF       = segment number to which the prescribed motion
!C                     is relative to
!C       NUM_VTAB    = number of time points of the vehicle acceleration
!C                     profile for options 2, 3, 4
!C       VNORMAL     = normal vector for the direction of the acceleration
!C                     pulse for options 1 and 2
!C       LIN_DATA    = linear vector acceleration time profile for the vehicle
!C       ANG_DATA    = angular vector acceleration time profile for the vehicle
!C       VOMEGA      = frequency for the half-sine wave option
!C       VINIT_TIME  = initial time of the vehicle acceleration profile for
!C                     options 2, 3, 4
!C       VDELTA_TIME = time interval of the vehicle acceleration profile data
!C                     for options 2, 3, 4
!C       TIMEV       = time duration of the half-sine wave deceleration for
!C                     option 1 of the vehicle input
!C
      TYPE VEHICLE
       CHARACTER ( LEN = 4,  KIND = ICHAR_STD )    VNAME          ! was VEH
       CHARACTER ( LEN = 80, KIND = ICHAR_STD )    VTITLE         ! was VPSTTL
       INTEGER   ( KIND = INTEGER_STD )   IVFLG                   ! was IVREF(1,)
       INTEGER   ( KIND = INTEGER_STD )   IVSEG                   ! was IVREF(2,)
       INTEGER   ( KIND = INTEGER_STD )   IVREF                   ! was IVREF(3,)
       INTEGER   ( KIND = INTEGER_STD )   NUM_VTAB                ! was NVTAB(6)
       REAL      ( KIND = IREAL_HIGH )    VNORMAL(3)              ! was AXV(3,6)
       REAL      ( KIND = IREAL_HIGH )    LIN_DATA(3,MAXVDT)      ! was VATAB(1-3,,)
       REAL      ( KIND = IREAL_HIGH )    ANG_DATA(3,MAXVDT)      ! was VATAB(4-6,,)
       REAL      ( KIND = IREAL_HIGH )    VOMEGA                  ! was OMEGV(6)
       REAL      ( KIND = IREAL_HIGH )    VINIT_TIME              ! was VT0(6)
       REAL      ( KIND = IREAL_HIGH )    VDELTA_TIME             ! was VTD(6)
       REAL      ( KIND = IREAL_HIGH )    TIMEV                   ! was TIMEV(6)
      END TYPE VEHICLE
!C
      TYPE ( VEHICLE )  VEH
      DIMENSION         VEH(MAXVEH)
!C
!C-------------------------------------------------------------------------------
!C     Define the joint actuator structure:
!C
!C       ACT_JNT             = joint number the actuator is associated with
!C       BASE_SEG            = segment number to be the base segment for the
!C                             actuator
!C       TARGET_ANGLE_FUNCT = function number for the target joint angle for
!C                             joint ACT_JNT, which the desired angle between
!C                             the two joint coordinate systems associated with
!C                             joint ACT_JNT
!C       PROPOR_FUNCT        = function number for the proportional gain control
!C                             variable in the actuator torque control equation
!C       DERIV_FUNCT         = function number for the derivative gain control
!C                             variable in the actuator torque control equation
```

```
!C      INTEGRAL_FUNCT      = function number for the integral gain control
!C                            variable in the actuator torque control equation
!C      TOR_AXIS            = vector for the torque axis of the actuator joint,
!C                            which is equal to the pin axis of the joint the
!C                            actuator is associated with
!C      ACT_TORQ            = magnitude of actuator torque based on PID control
!C      PROP_TORQ           = proportional component of actuator torque
!C      DERIV_TORQ          = derivative component of actuator torque
!C      INTEGRAL_TORQ       = integral component of actuator torque
!C      ACT_JNT_ANGLE       = angle of joint associated with actuator
!C      ACT_JNT_VEL         = velocity of joint associated with actuator
!C
       TYPE ACTUATOR
        INTEGER    ( KIND = INTEGER_STD )   ACT_JNT              ! was NRJNT
        INTEGER    ( KIND = INTEGER_STD )   BASE_SEG             ! was NRS
        INTEGER    ( KIND = INTEGER_STD )   TARGET_ANGLE_FUNCT   ! was NRF(1,)
        INTEGER    ( KIND = INTEGER_STD )   PROPOR_FUNCT         ! was NRF(2,)
        INTEGER    ( KIND = INTEGER_STD )   DERIV_FUNCT          ! was NRF(3,)

        INTEGER    ( KIND = INTEGER_STD )   INTEGRAL_FUNCT       ! was NRF(4,)
        REAL       ( KIND = IREAL_HIGH )    TOR_AXIS(3)          ! was QRU(1-3,)
        REAL       ( KIND = IREAL_HIGH )    ACT_TORQ             ! was TORQUE(1,)
        REAL       ( KIND = IREAL_HIGH )    PROP_TORQ            ! was TORQUE(2,)
        REAL       ( KIND = IREAL_HIGH )    DERIV_TORQ           ! was TORQUE(3,)
        REAL       ( KIND = IREAL_HIGH )    INTEGRAL_TORQ        ! was TORQUE(4,)
        REAL       ( KIND = IREAL_HIGH )    ACT_JNT_ANGLE        ! was TORQUE(5,)
        REAL       ( KIND = IREAL_HIGH )    ACT_JNT_VEL          ! was TORQUE(6,)
       END TYPE ACTUATOR
!C
       TYPE ( ACTUATOR )  ACT
       DIMENSION          ACT(MAXJNT)
!C
!C
!C*****************************************************************************
!C    Define the global variables.
!C*****************************************************************************
!C
!C    /ABDATA/
!C
       INTEGER  ( KIND = INTEGER_STD)  IFULL
       DIMENSION  IFULL(6)
!C
       REAL ( KIND = IREAL_HIGH)
      &                  ZDEP, DBR, DPVCTR, DEPLOY, AB, B, ZR, BFB,
      &                  DRR, VBAGG, VSCS, SPRK, CK, CMASS, CYMIN,
      &                  CYMOUT, BAGPV, PD, VBAG, VOLBP, PCYV,
      &                  PCYMIN, PVBAG, TV1, TV2, SWITCH, PYMOUT,
      &                  SCALEX, PREVT
       DIMENSION  ZDEP(3,5), DBR(3,3,5), DPVCTR(3,5), DEPLOY(3,5),
      &           AB(3,5), B(9,4,5), ZR(3,4,5), BFB(3,4,5), DRR(9,4,5),
      &           VBAGG(5), VSCS(5), SPRK(5), CK(5), CMASS(5), CYMIN(5),
      &          CYMOUT(5), BAGPV(5), PD(5), VBAG(5), VOLBP(5),
      &         PCYV(5), PCYMIN(5), PVBAG(5), TV1(3,4,5), TV2(3,10,5),
      &          SWITCH(5), PYMOUT(5), SCALEX(5)
!C
!C     COMMON /ABDATA/  ZDEP, DBR, DPVCTR, DEPLOY, AB, B, ZR, BFB,
!C    *                 DRR, VBAGG, VSCS, SPRK, CK, CMASS, CYMIN,
!C    *                 CYMOUT, BAGPV, PD, VBAG, VOLBP, PCYV,
!C    *                 PCYMIN, PVBAG, TV1, TV2, SWITCH, PYMOUT,
!C    *                 SCALE, PREVT, IFULL
!C
!C*****************************************************************************
!C
!C     /ACTFR/
!C
      INTEGER  ( KIND = INTEGER_STD )  NRTORQ
!C
!C      COMMON /ACTFR/  QRU, TORQUE, NRTORQ, NRJNT, NRS
!C
!C*****************************************************************************
!C
```

```
!C     /ACTFR1/
!C
      REAL   ( KIND = IREAL_HIGH )  ACT_TIME_PREV, ACT_THETA_CUR,
     &                                              ACT_THETA_PREV
!C
!C      COMMON /ACTFR1/  NRF, TIMEI, THETAI
!C
!C*********************************************************************
!C
!C     /BAGDIM/
!C
!C    This common block was in Subroutine U1ASCD in V.1, but not in
!C     BLOCK DATA.
!C
      INTEGER   ( KIND = INTEGER_STD )
     &           IBMODL, IBSHP, IBATT, IBFOLD, IBDPLY, IBMOV, IBSEG,
     &           NBSEG, NDPLT, NDPLPT
      DIMENSION  IBMODL(MAXBAG), IBSHP(MAXBAG), IBATT(MAXBAG),
     &           IBFOLD(MAXBAG), IBDPLY(MAXBAG), IBMOV(MAXBAG),
     &           IBSEG(MAXBAG), NBSEG(MAXBAG), NDPLT(MAXBAG),
     &           NDPLPT(MAXBDT,MAXBAG)
!C
      REAL   ( KIND = IREAL_HIGH )
     &                 BAGSHP, BAGAXS, DPLX, DPLA, DPLTIM, ATTDIM,
     &                 FOLDX, DPLYT, DPLYPT, PNL1, PNL2, PNL3,
     &                 BLEN, BAREA, BVOL, BWGT
      DIMENSION        BAGSHP(3,MAXBSH,MAXBAG), BAGAXS(MAXBSH,MAXBAG),
     &                 DPLX(3,MAXBAG), DPLA(3,MAXBAG), DPLTIM(MAXBAG),
     &                 ATTDIM(2,MAXBAG), FOLDX(4,MAXBAG),
     &                 DPLYT(MAXBDT,MAXBAG), DPLYPT(MAXBDP,MAXBAG),
     &                 PNL1(3), PNL2(3), PNL3(3), BLEN(MAXBAG),
     &                 BAREA(MAXBAG), BVOL(MAXBAG), BWGT(MAXBAG)
!C
!C*******
!C
!C    /CDH10C/
!C
      INTEGER   ( KIND = INTEGER_STD )  ISEQ, IDCG
      DIMENSION  ISEQ(3,5), IDCG(5)
!C
      REAL   ( KIND = IREAL_HIGH )  ORIGIN, XYZANG
      DIMENSION  ORIGIN(3,5), XYZANG(3,5)
!C
!C     COMMON /CDH10C/ ORIGIN, XYZANG, ISEQ, IDCG
!C
!C*********************************************************************
!C
!C    /CDINT/
!C
!C    NOTE:  FF REPLACES F.
!C
      INTEGER   ( KIND = INTEGER_STD )  ICNT, IDBL, IFLAG
!C
      REAL   ( KIND = IREAL_HIGH )
     &                 UU, GH, E, FF, GG, Y, U, H, HPRINT, TSAVE,
     &                 TPRINT, TSTART
      DIMENSION  UU(4), GH(3,4), E(3,3*MAXEQN), FF(5,3*MAXEQN),
     &           GG(5,3*MAXEQN), Y(5,3*MAXEQN), U(5,3*MAXEQN)
!C
!C     COMMON/CDINT/  UU, GH, E, FF, GG, Y, U, H, HPRINT, TSAVE,
!C    *               TPRINT, TSTART, ICNT, IDBL, IFLAG
!C
!C*********************************************************************
!C
!C    /CEULER/
!C
      INTEGER   ( KIND = INTEGER_STD )  IEULER
      DIMENSION  IEULER(MAXJNT)
!C
      REAL   ( KIND = IREAL_HIGH )  HIR, ANG, ANGD, FE, TQE
      DIMENSION  HIR(3,3,3*MAXJNT), ANG(3,MAXJNT), ANGD(3,MAXJNT),
```

```
     &            FE(3,MAXJNT), TQE(3,MAXJNT)
!C
!C    COMMON/CEULER/ IEULER, HIR, ANG, ANGD, FE, TQE, CONST
!C
!C*****************************************************************************
!C
!C    /CMATRX/
!C
      REAL   ( KIND = IREAL_HIGH )
     &            V1, V2, V3, B12, A22, WJ, A11
      DIMENSION  V1(3,MAXJNT), V2(3,MAXJNT), V3(3,MAXCST),
     &            B12(3,3,2*MAXJNT), A22(3,3,2*MAXJNT),
     &            WJ(MAXJNT), A11(3,3,MAXJNT)
!C
!C    COMMON /CMATRX/ V1, V2, V3, B12, A22, F, TQ, WJ, A11
!C
!C*****************************************************************************
!C
!C    /CNSNTS/
!C
      REAL   ( KIND = IREAL_HIGH )
     &                  PI, RADIAN, G, EPS, GRAVTY, TWOPI
      DIMENSION         EPS(24), GRAVTY(3)
!C
      CHARACTER ( LEN = 8, KIND = ICHAR_STD )  UNITL, UNITM, UNITT
!C
!C    COMMON /CNSNTS/ PI, RADIAN, G, THIRD, EPS,
!C    *                UNITL, UNITM, UNITT, GRAVTY, TWOPI
!C    Note:  added the constant ZERO to this grouping.
!C
!C*****************************************************************************
!C
!C    /CNTSRF/
!C
      REAL   ( KIND = IREAL_HIGH )  PL, BELT, TPTS, BD, BELT_FORCE
      DIMENSION  PL(24,MAXPLN), BELT(20,MAX_NUM_BELTS),
     &            TPTS(6,MAX_NUM_BELTS), BD(24,MAXELP),
     &            BELT_FORCE(4,MAX_NUM_BELTS)
!C
!C    COMMON /CNTSRF/ PL, BELT, TPTS, BD
!C
!C*****************************************************************************
!C
!C    /COMAIN/
!C
      INTEGER   ( KIND = INTEGER_STD )  ISTEP, NSTEPS, NDINT, NEQ
!C
      REAL   ( KIND = IREAL_HIGH )  VAR, DER, DT, H0, HMAX, HMIN
      DIMENSION  VAR(3*MAXEQN), DER(3*MAXEQN)
!C
!C     COMMON /COMAIN/  VAR, DER, DT, H0, HMAX, HMIN, ISTEP, NSTEPS,
!C    *                 NDINT, NEQ
!C
!C*****************************************************************************
!C
!C    /CONTRL/
!C
      INTEGER   ( KIND = INTEGER_STD )
     &          NSEG, NJNT, NPL, NBLT, NBAG, NVEH, NGRND,
     &          NS, NQ, NSD, NFLX, NHRNSS, NWINDF, NJNTF, NPRT,
     &          NPG
      DIMENSION  NPRT(36)
!C

      REAL   ( KIND = IREAL_HIGH )  TIME
!C
!C    COMMON /CONTRL/ TIME, NSEG, NJNT, NPL, NBLT, NBAG, NVEH, NGRND,
!C   *                NS, NQ, NSD, NFLX, NHRNSS, NWINDF, NJNTF, NPRT,
!C   *                NPG
!C
!C*****************************************************************************
```

```
!C
!C    /COUT/
!C
!C    Was in Subroutine FINPUT, HEDING in Release V.1, but was not
!C     included in the BLOCK DATA routine.
!C
      INTEGER  ( KIND = INTEGER_STD )   NOUTPS, NOUTSS
      DIMENSION  NOUTPS(MAXPSF), NOUTSS(MAXSSF)
!C
!C      COMMON/COUT/ NOUTPS(MAXPSF), NOUTSS(MAXSSF)
!C
!C
!C*****************************************************************************
!C
!C      /COUTFMT/
!C
!C    This common was in Subroutine U1ASCD in V.1, but not in the BLOCK
!C     DATA.
!C
      INTEGER  ( KIND = INTEGER_STD )
     &          IBAGFM, IBAGTY, IHARFM, IHARTY, IRGDFM, IRGDTY,
     &          ITIMFM, ITIMTY

!C
!C      COMMON/COUTFMT/  ITIMTY,ITIMFM,IRGDTY,IRGDFM,
!C     *                 IBAGTY,IBAGFM,IHARTY,IHARFM
!C
!C*****************************************************************************
!C
!C      /COUTN/
!C
!C    This common was in Subroutine U1ASCD in V.1, but not in the BLOCK
!C     DATA statement.
!C
      INTEGER  ( KIND = INTEGER_STD )  NTOBLT, NTOPTS, LDUM, LPREV
!C
!C      COMMON/COUTN/    NTOBLT,NTOPTS,LDUM,LPREV
!C
!C
!C*****************************************************************************
!C
!C    /CSTRNT/
!C
      INTEGER    ( KIND = INTEGER_STD )  KQ1, KQ2, KQTYPE
      DIMENSION  KQ1(MAXCST), KQ2(MAXCST), KQTYPE(MAXCST)
!C
      REAL  ( KIND = IREAL_HIGH )
     &                  A13, A23, B31, B32, HHT, RK1, RK2, QQ, TQQ,
     &                  RQQ, HQQ, SQQ, CFQQ
      DIMENSION  A13(3,3,I_2*MAXCST), A23(3,3,I_2*MAXCST),
     &           B31(3,3,I_2*MAXCST), B32(3,3,I_2*MAXCST),
     &           HHT(3,3,MAXCST), RK1(3,MAXCST), RK2(3,MAXCST),
     &           QQ(3,MAXCST), TQQ(3,MAXCST), RQQ(3,MAXCST),
     &           HQQ(3,MAXCST), SQQ(MAXCST), CFQQ(MAXCST)
!C
!C     COMMON /CSTRNT/ A13, A23, B31, B32, HHT, RK1, RK2, QQ, TQQ,
!C    *                RQQ, HQQ, SQQ, CFQQ, KQ1, KQ2, KQTYPE
!C
!C*****************************************************************************
!C
!C    /CYDATA/
!C
      REAL ( KIND = IREAL_HIGH )
     &                CYTD, CYPA, CYSP, CYT0, CYV0, CYCD, CYK, CYR,
     &                 CYAT, CYPV, CYCD0, CYP0, CYSS, CYL0, CYC, CYA0,
     &                 CYRHO0, CYVMAX, CYORFC, CYRHO, CYT, CYP, CYV
      DIMENSION   CYTD(MAXBAG), CYPA(MAXBAG), CYSP(MAXBAG),
     &            CYT0(MAXBAG), CYV0(MAXBAG), CYCD(MAXBAG),
     &            CYK(MAXBAG), CYR(MAXBAG), CYAT(MAXBAG),
     &            CYPV(MAXBAG), CYCD0(MAXBAG), CYP0(MAXBAG),
     &            CYSS(MAXBAG), CYL0(MAXBAG), CYC(MAXBAG),
```

```
     &             CYRHO0(MAXBAG), CYVMAX(MAXBAG), CYORFC(MAXBAG),
     &             CYRHO(MAXBAG), CYT(MAXBAG), CYP(MAXBAG),
     &             CYV(MAXBAG), CYA0(MAXBAG)
!C
!C      COMMON /CYDATA/   CYTD, CYPA, CYSP, CYT0, CYV0, CYCD, CYK, CYR,
!C     *                  CYAT, CYPV, CYCD0, CYA0, CYP0, CYSS, CYL0, CYC,
!C     *                  CYRHO0, CYVMAX, CYORFC, CYRHO, CYT, CYP, CYV
!C
!C*******************************************************************************
!C
!C    /DAMPER/
!C
      INTEGER  ( KIND = INTEGER_STD )  MSDM, MSDN
      DIMENSION  MSDM(MAX_NUM_DAMPERS), MSDN(MAX_NUM_DAMPERS)
!C
      REAL ( KIND = IREAL_HIGH )  APSDM, APSDN, ASD, DAMP_FORCE
      DIMENSION  APSDM(3,MAX_NUM_DAMPERS), APSDN(3,MAX_NUM_DAMPERS),
     &           ASD(5,MAX_NUM_DAMPERS), DAMP_FORCE(4,MAX_NUM_DAMPERS)
!C
!C    COMMON/DAMPER/ APSDM, APSDN, ASD, MSDM, MSDN
!C
!C*******************************************************************************
!C
!C    /DESCRP/
!C
      INTEGER  ( KIND = INTEGER_STD ) IGLOB, JOINTF
      DIMENSION  IGLOB(MAXJNT), JOINTF(3,MAXJNT)
!C
      REAL  ( KIND = IREAL_HIGH )
     &           HT, SPRING, VISC
      DIMENSION  HT(3,3,2*MAXJNT), SPRING(5,3*MAXJNT),
     &           VISC(7,3*MAXJNT)
!C
!C     COMMON /DESCRP/ PHI, W, RW, SR, HA, HB, RPHI, HT, SPRING,
!C    *                VISC, JNT, IPIN, ISING, IGLOB, JOINTF
!C
!C*******************************************************************************
!C
!C    /FILEN/
!C
      CHARACTER ( LEN = 32, KIND = ICHAR_STD )  INFIL, OUTFIL
      CHARACTER ( LEN = 80, KIND = ICHAR_STD )  WORK_DIRECTORY
!C

!C    COMMON/FILEN/  OUTFIL
!C
!C*******************************************************************************
!C
!C    /FLXBLE/
!C
      INTEGER  ( KIND = INTEGER_STD )  NFLEX
      DIMENSION  NFLEX(3,MAXFLX)
!C
      REAL  ( KIND = IREAL_HIGH )   HF, B42, V4
      DIMENSION  HF(4,12,MAXFLX), B42(3,3,I_3*MAXFLX), V4(3,MAXFLX)
!C
!C    COMMON /FLXBLE/  HF, B42, V4, NFLEX
!C
!C*******************************************************************************
!C
!C    /FORCES/
!C
!C    NBGSF         = number of airbag-segment tabular time histories
!C    NBSF          = number of harness belt tabular time histories
!C    NPANEL        =
!C    NPSF          = number of plane-segment tabular time histories
!C    NSSF          = number of segment-segment tabular time histories
!C
!C    BAGSF         = array containing airbag-segment tabular time history data
!C    HARNESS_FORCE = array containing harness belt tabular time history data
!C    PRJNT         = array containing joint tabular time history data
```

```
!C    PSF            = array containing plane-segment tabular time history data
!C    SSF            = array containing segment-segment tabular time history data
!C
      INTEGER  ( KIND = INTEGER_STD )   NPANEL, NPSF, NBSF, NSSF, NBGSF
      DIMENSION  NPANEL(MAXBAG)
!C
      REAL  ( KIND = IREAL_HIGH )   PSF, HARNESS_FORCE, SSF, BAGSF,
     &                              PRJNT
      DIMENSION  PSF(7,MAXPSF), HARNESS_FORCE(4,MAXHBLT),
     &           SSF(10, MAXSSF), BAGSF(3,20), PRJNT(7,MAXJNT)
!C
!C     COMMON /FORCES/ PSF, BSF, SSF, BAGSF, PRJNT, NPANEL, NPSF, NBSF,
!C    *                NSSF, NBGSF
!C


!C*****************************************************************************
!C
!C    /HBPTRB/
!C
!C    HRN_EPSDEL  = maximum strain convergence criterion used for balancing
!C                  the harness belts
!C    HRN_MAX_ITR = maximum number of iterations used to meet the maximum
!C                  strain convergence criterion for the harness belts
!C
      INTEGER  ( KIND = INTEGER_STD )  HRN_MAX_ITR
!C
      REAL  ( KIND = IREAL_HIGH )   HRN_EPSDEL
!C
!C    COMMON /HBPTRB/  EPSDEL, MAXITR
!C
!C*****************************************************************************
!C
!C    /HRNESS/
!C
      INTEGER  ( KIND = INTEGER_STD )
     &           IBAR, NL, NPTSPB, NPTPLY, NTHRNS, NBLTPH, KEEP
      DIMENSION  IBAR(5,MAXHPT), NL(2,MAXHPT), NPTSPB(MAXHBLT),
     &           NPTPLY(MAXHBLT), NTHRNS(MAXHBLT,25), NBLTPH(MAXHRN),
     &           KEEP(MAXHPT)
!C
      REAL  ( KIND = IREAL_HIGH )  BAR, BB, BBDOT, PLOSS, XLONG, HTIME
      DIMENSION  BAR(15,MAXHPT), BB(MAXHPT), BBDOT(MAXHPT),
     &           PLOSS(2,MAXHPT), XLONG(MAXHBLT), HTIME(2)
!C
!C     COMMON /HRNESS/ BAR, BB, BBDOT, PLOSS, XLONG, HTIME, IBAR, NL,
!C    *                NPTSPB, NPTPLY, NTHRNS, NBLTPH, KEEP
!C
!C*****************************************************************************
!C
!C    /INTEST/
!C
      CHARACTER ( LEN = 4, KIND = ICHAR_STD )  SEGT
      DIMENSION      SEGT(4*MAXSEG)
!C
      REAL  ( KIND = IREAL_HIGH )  XTEST
      DIMENSION  XTEST(3,4*MAXSEG)
!C
!C    Create single dimensioned versions of the XTEST and REGT arrays
!C     for Subroutines DINTG and PDAUX.
!C
      CHARACTER ( LEN = 8, KIND = ICHAR_STD )  REGT_SNGL
      DIMENSION      REGT_SNGL(4*MAXSEG)
!C
      REAL  ( KIND = IREAL_HIGH )  XTEST_SNGL
      DIMENSION  XTEST_SNGL(3*4*MAXSEG)
!C
      EQUIVALENCE  ( XTEST, XTEST_SNGL )
!C
!C    COMMON /INTEST/  SGTEST, XTEST, SEGT, REGT
!C
```

```
!C*****************************************************************************
!C
!C    /JBARTZ/
!C
      INTEGER   ( KIND = INTEGER_STD )
     &          MNPL, MNBLT, MNSEG, MNBAG, MPL, MBLT, MSEG, MBAG,
     &          NTPL, NTBLT, NTSEG
      DIMENSION  MNPL(MAXPLN), MNBLT(MAX_NUM_BELTS), MNSEG(MAXSEG),
     &           MNBAG(6), MPL(3,5,MAXPLN), MBLT(3,5,MAX_NUM_BELTS),
     &           MSEG(3,5,MAXSEG), MBAG(3,10,6), NTPL(5,MAXPLN),
     &           NTBLT(5,MAX_NUM_BELTS), NTSEG(5,MAXSEG)
!C
!C     COMMON/JBARTZ/ MNPL, MNBLT, MNSEG, MNBAG, MPL, MBLT, MSEG,
!C    *               MBAG, NTPL, NTBLT, NTSEG
!C
!C*****************************************************************************
!C
!C    /RSAVE/
!C
      INTEGER   ( KIND = INTEGER_STD )
     &           NSG, MSG, MCG, MCGIN, KREF, MJS
      DIMENSION  NSG(10), MSG(MAX_HCARD_TTH,10),
     &           MCGIN(24,MAX_TOTAL_BODY),
     &           KREF(MAX_HCARD_TTH,10), MJS(MAX_HCARD_TTH,2)
!C
      REAL   ( KIND = IREAL_HIGH )  XSG
      DIMENSION  XSG(3,MAX_HCARD_TTH,3)
!C
!C    COMMON /RSAVE/  XSG, DPMI, LPMI, NSG, MSG, MCG, MCGIN, KREF, MJS
!C
!C*****************************************************************************
!C
!C    /SGMNTS/
!C
       INTEGER    ( KIND = INTEGER_STD )   SYMMETRY          ! was NSYM
       DIMENSION                           SYMMETRY(MAXSEG)
!C
!C    COMMON /SGMNTS/ D, WMEG, WMEGD, U1, U2, SEGLP, SEGLV, SEGLA, NSYM
!C
!C*****************************************************************************
!C
!C    /TABLES/
!C
      INTEGER   ( KIND = INTEGER_STD )
     &                    MXNTB, MXTB1, MXTB2, NTI, NTAB
      DIMENSION  NTI(MAX_FUNC), NTAB(MAXNTB)
!C
      REAL   ( KIND = IREAL_HIGH )   TAB
      DIMENSION  TAB(MAXTAB)
!C
!C    COMMON /TABLES/ MXNTI, MXNTB, MXTB1, MXTB2, NTI, NTAB, TAB
!C
!C*****************************************************************************
!C
!C     /TEMPVI/
!C
      INTEGER    ( KIND = INTEGER_STD )  JSTOP
      DIMENSION  JSTOP(4,2,MAXJNT)
!C
      REAL   ( KIND = IREAL_HIGH  )   CREST, TTI, R1I, R2I
      DIMENSION  TTI(3), R1I(3), R2I(3)
!C
!C    COMMON /TEMPVI/  CREST, TTI, R1I, R2I, JSTOP
!C
!C*****************************************************************************
!C
!C    /TITLES/
!C
      CHARACTER   ( LEN = 12,   KIND = ICHAR_STD )   DATE
      CHARACTER   ( LEN = 20,   KIND = ICHAR_STD )
     &                  BAGTTL, BDYTTL, BLTTTL, PLTTL
```

```
      DIMENSION          BAGTTL(6), BLTTTL(MAX_NUM_BELTS), PLTTL(MAXPLN)
      CHARACTER   ( LEN = 80,   KIND = ICHAR_STD )    VPSTTL
      CHARACTER   ( LEN = 160, KIND = ICHAR_STD)    COMENT
!C
!C      REAL  DATE, COMENT, VPSTTL, BDYTTL, BLTTTL, PLTTL, BAGTTL,
!C     &      SEG, JOINT
!C      DIMENSION  DATE(3), COMENT(40), VPSTTL(20), BDYTTL(5),
!C     &           BLTTTL(5,8), PLTTL(5,MAXPLN), BAGTTL(5,6),
!C     &          SEG(MAXSEG), JOINT(MAXJNT)
!C
!C
!C     COMMON /TITLES/ DATE, COMENT, VPSTTL, BDYTTL, BLTTTL, PLTTL,
!C    *                BAGTTL, SEG, JOINT, CGS, JS
!C
!C*****************************************************************************
!C
!C     /TMPVS2/
!C
!C     COMMON/TMPVS2/ FREE
!C
!C
!C*****************************************************************************
!C
!C     /VPOSTN/
!C
!C     NUMVEH = number of vehicles
!C
      INTEGER   ( KIND = INTEGER_STD ) NUMVEH
!C
!C     COMMON /VPOSTN/ ZPLT, SPLT, AXV, VATAB, VT0, VDT, TIMEV, OMEGV,
!C    *                NVTAB, NUMVEH, IVREF
!C
!C*****************************************************************************
!C
!C     /WINDFR/
!C
      INTEGER   ( KIND = INTEGER_STD )
     &           IWIND, MWSEG, NFVSEG, NFVNT, MOWSEG, MOWELP,
     &           NFORCE
      DIMENSION  IWIND(MAXSEG), MWSEG(7,MAXSEG),
     &           NFVSEG(MAX_FOR_TORQ),
     &           NFVNT(MAX_FOR_TORQ), MOWSEG(MAXSEG,MAXSEG),
     &           MOWELP(MAXSEG,MAXSEG)
!C
      REAL   ( KIND = IREAL_HIGH )    WTIME, QFU, QFV, WF
      DIMENSION  WTIME(MAXSEG), QFU(3,MAX_FOR_TORQ),
     &           QFV(3,MAX_FOR_TORQ), WF(3,MAXSEG)
!C
!C      COMMON /WINDFR/  WTIME, QFU, QFV, WF, IWIND, MWSEG, NFVSEG,
!C     *                 NFVNT, MOWSEG
!C
!C*****************************************************************************
!C
!C     /XTRA/
!C
      INTEGER   ( KIND = INTEGER_STD )    NN, NELP, MELL
      DIMENSION  MELL(MAXELP)
!C
      REAL   ( KIND = IREAL_STD )    P1S, P2S, P3S, P4S
      DIMENSION  P1S(3,MAXELP), P2S(3,MAXELP), P3S(3,MAXELP),

     &           P4S(3,MAXELP)
!C
!C     COMMON /XTRA/  NN, NELP, MELL, P1S, P2S, P3S, P4S
!C
!C
!C*****************************************************************************
!C*****************************************************************************
!C
!C     This module contains variables that are shared amongst the
!C      following Airbag subroutines:
```

```
!C
!C       /AIRBAG_TEMPVS/
!C
      REAL   ( KIND = IREAL_STD )
     &                  TMP, TMP1, TORQ, FORCE, TORA,
     &                  TQB, FRB, VOL, DELF, VOLP, FRA
      DIMENSION TMP(9), TMP1(3),TORQ(3), FORCE(3,5), TORA(3,5),
     &          TQB(3,10), FRB(3,10), VOL(10), DELF(3),
     &          VOLP(4,5), FRA(4,5)
!C
!C     COMMON/TEMPVS/ TMP, TMP1, TORQ, FORCE, TORA,
!C    *                  TQB, FRB, VOL, DELF, VOLP, FRAC
!C
!C     COMMON/TEMPVS/ TMP(9),TMP1(3),TORQ(3),FORCE(3,5),TORA(3,5),
!C    *            TQB(3,10),FRB(3,10),VOL(10),DELF(3),VOLP(4,5),FRA(4,5)
!C     NOTE: THIS COMMON/TEMPVS/ IS SHARED BY AIRBAG AND AIRBGG.
!C
!C*****************************************************************************
!C
!C     /BELT_TEMPVS/
!C
!C     This module replaces the /TEMPVS/ shared by Subroutines
!C      BELTRT and BELTG
!C
!C     NOTE: BELTRT AND BELTG SHARE FIRST PART OF TEMPVS
!C
      REAL   ( KIND = IREAL_HIGH )
     &           APA, UVA, DLGA, UAA, APB, UVB, DLGB, UBB
      DIMENSION  APA(3), UVA(3), APB(3), UVB(3)
!C
!C      COMMON/TEMPVS/ APA(3),UVA(3),DLGA,UAA,APB(3),UVB(3),DLGB,UBB
!C     *              ,TA(3),TB(3),TC(3),UP(3),B(3)
!C     *              ,UC(3),AX(3),XE(3),BX(3),ACA(3),ACB(3)
!C
!C*****************************************************************************
!C
!C      /CINPUT_TEMPVS/
!C
!C    These variables were in a /TEMPVS/ that was shared by Subroutines
!C      CINPUT, FDINIT, FINPUT, HINPUT.  NF is renamed to NF_FUNCT and
!C      MS is renamed to MS_SEG.
!C
      INTEGER  ( KIND = INTEGER_STD )  NF_FUNCT, MS_SEG
      DIMENSION  NF_FUNCT(5), MS_SEG(3)
!C
      CHARACTER ( LEN = 4,  KIND = ICHAR_STD )   KTITLE
      CHARACTER ( LEN = 20, KIND = ICHAR_STD )   FUNC_TITLE
      DIMENSION       FUNC_TITLE(MAX_FUNC+1), KTITLE(31)
!C
!C    NOTE: THIS IS SHARED BY SUBS CINPUT, FINPUT, HINPUT AND FDINIT.
!C          also used by Subroutine KINPUT
!C
!C    COMMON/TEMPVS/ JTITLE(5,51),NF(5),MS(3),KTITLE(31)
!C    REAL JTITLE,KTITLE
!C
!C*****************************************************************************
!C
!C       /DAUX_TEMPVS/
!C
!C    Note: this /TEMPVS/ is shared by DAUX11, DAUX12, DAUX22,
!C          DAUX31, DAUX32, and DAUX33.
!C
      INTEGER  ( KIND = INTEGER_STD )  IJK, IJ, NQ2S
      DIMENSION  IJK(MAXRHS,MAXRHS)
!C
      REAL  ( KIND = IREAL_HIGH )   C, RHS
      DIMENSION  C(3,3,MAXCMX), RHS(3,MAXRHS)
!C
!C    LOGICAL*1 FREE
!C    COMMON/TEMPVS/ C(3,3,MAXCMX),RHS(3,MAXRHS),IJK(MAXRHS,MAXRHS),
!C   *               IJ,NQ2S
```

```
!C    COMMON/TMPVS2/ FREE(MAXJNT)
!C
!C    /TEMPVS/
!C
!C    INTEGER  JTMPVS
!C    DIMENSION  JTMPVS(MAXTMP)
!C
!C    COMMON /TEMPVS/ JTMPVS
!C
!C*******************************************************************************
!C
!C     /HEDING_TEMPVS/
!C
!C     Note: Subroutines POSTPR, HEDING, and HEDINGX shared this
!C           COMMON /TEMPVS/.
!C
!C     SEE COMMENT IN POSTPR ABOUT FIRST DIMENSION OF PLDATA.
!C
      INTEGER  ( KIND = INTEGER_STD )    NOPL, MOPL, M1PL, M2PL
      DIMENSION  NOPL(MHEDNG), MOPL(MHEDNG), M1PL(MHEDNG), M2PL(MHEDNG)
!C
      REAL  ( KIND = IREAL_HIGH )   TDATA
      DIMENSION          TDATA(14,MAX_NUM_TTH)
!C
      REAL  ( KIND = IREAL_STD )   USEC, ZTTH
      DIMENSION  USEC(MAX_LN_PPAGE), ZTTH(14,MAX_LN_PPAGE,MAX_NUM_TTH)
!C
      CHARACTER ( LEN = 4, KIND = ICHAR_STD )  HEAD
      DIMENSION      HEAD(20)
!C
!C      As was in Subroutine HEDING.
!C      COMMON/TEMPVS/ TDATA(14,65),HEAD(20),NOPL(MHEDNG),MOPL(MHEDNG),
!C     *               M1PL(MHEDNG),USEC(45),ZTTH(14,45,65),
!C     *               M2PL(MHEDNG)
!C
!C      As was in Subroutine POSTPR.
!C      COMMON/TEMPVS/ TDATA(14,65),HEDATA(3*MHEDNG+20),
!C     *               USEC(45),ZTTH(14,45,65),M2PL(MHEDNG)
!C
!C*******************************************************************************
!C
!C       /HRN_TEMPVS/
!C
!C    The variables were in a /TEMPVS/ that was shared by Subroutines
!C     HPTURB, HBPLAY, HBELT, and HSETC.  The following variables
!C     were renamed:
!C
!C          B  to B_HRN     E   to E_HRN        FP   to FP_HRN
!C          S  to S_HRN     FCE to FCE_HRN      RHS  to RHS_HRN
!C          T  to T_HRN         FR  to FR_HRN       C    to C_HRN
!C          R  to R_HRN     ZR  to ZR_HRN       IJK  to IJK_HRN
!C          V  to V_HRN     TR  to TR_HRN       NOLD to NOLD_HRN
!C          T1 to T1_HRN    BL  to BL_HRN
!C          T2 to T2_HRN    FB  to FB_HRN
!C
      INTEGER  ( KIND = INTEGER_STD )  IJK_HRN, NOLD_HRN
      DIMENSION  IJK_HRN(54,54), NOLD_HRN(2,MAXHPT)
!C
      REAL  ( KIND = IREAL_HIGH )
     &                  B_HRN, S_HRN, T_HRN, R_HRN, V_HRN, T1_HRN,
     &                  T2_HRN, E_HRN, EDOT, FCE_HRN, FR_HRN, ZR_HRN,
     &                  TR_HRN, U_HRN, PTLOSS, BL_HRN, FB_HRN, FP_HRN,
     &                  OLDBB, RHS_HRN, C_HRN
      DIMENSION  B_HRN(3,3,3), S_HRN(3,3), T_HRN(3), R_HRN(3),
     &           V_HRN(3), T1_HRN(3), T2_HRN(3), E_HRN(3,3,MAXHPH),
     &           EDOT(3,MAXHPH), FCE_HRN(3,MAXHPH), FR_HRN(3,MAXHPH),
     &           ZR_HRN(3,MAXHPH), TR_HRN(3,MAXHPH), U_HRN(3,MAXHPH),
     &           PTLOSS(2,MAXHPT), BL_HRN(MAXHPH), FB_HRN(MAXHPH),
     &           FP_HRN(MAXHPH), OLDBB(MAXHPT), RHS_HRN(3,54),
     &           C_HRN(3,3,200)
!C
```

```
!C     THIS COMMON/TEMPVS/ IS SHARED BY HPTURB, HBPLAY, HBELT AND HSETC.
!C
!C
!C     COMMON/TEMPVS/ B(3,3,3),S(3,3),T(3),R(3),V(3),T1(3),T2(3),
!C    *               E(3,3,50),EDOT(3,50),FCE(3,50),FR(3,50),ZR(3,50),
!C    *              TR(3,50),U(3,50),PTLOSS(2,100),BL(50),FB(50),FP(50),
!C    *            OLDBB(100),RHS(3,54),C(3,3,200),IJK(54,54),NOLD(2,100)
!C
!C    BLOSS and HLOSS are defined as equivalenced with C for Subroutine
!C     HPTURB.
!C
      REAL   ( KIND = IREAL_HIGH )    BLOSS, HLOSS
      DIMENSION          BLOSS(2,MAXHBLT), HLOSS(2,MAXHRN)
!C
      EQUIVALENCE   ( BLOSS(1,1), C_HRN(1,1, 1) ),
     &              ( HLOSS(1,1), C_HRN(1,1,10) )
!C
!C*****************************************************************************
!C
!C       /PLELP_TEMPVS/
!C
!C     This COMMON/TEMPVS/ is shared by PLEDG, PLELP, PLSEGF, and
!C                                      SEGSEG.
!C
      INTEGER   ( KIND = INTEGER_STD )   MCF_PLP, NCF_PLP
!C
      REAL   ( KIND = IREAL_HIGH )
     &                   AMR_PLP, DMNT_PLP, DMNWN_PLP, FM_PLP, PEN_DIST,
     &                   R_PLP, RLM_PLP, RLN_PLP,
     &                   RM_PLP, RMD_PLP, RN_PLP, RND_PLP,
     &                   T_PLP, TF_PLP, TH_PLP, TM_PLP, VR_PLP,
     &                   WNM_PLP, XH_PLP, XMM_PLP, XMN_PLP, XNC_PLP
      DIMENSION          R_PLP(3), DMNT_PLP(3,3), DMNWN_PLP(3),
     &                   RLM_PLP(3), RLN_PLP(3), RM_PLP(3),
     &                   RMD_PLP(3), RN_PLP(3), RND_PLP(3),
     &                   T_PLP(3), TH_PLP(3), TM_PLP(3), VR_PLP(3),
     &                   WNM_PLP(3), XH_PLP(3), XMM_PLP(3),
     &                   XMN_PLP(3), XNC_PLP(3)
!C
!C      As found in PLEDG; shared with PLELP-PLSEGF
!C        COMMON/TEMPVS/DMNT(3,3),DHNT(3,3),DUM1(18),TM(3),R(3),RM(3),
!C     X             DUM2(9),UP(3),VP(3),U(3),V(3),EU(3),EV(3),ET(3),
!C     X             A(2),B(2),CC(2),DUM4(12),TH(3),XH(3),RMD(3),RND(3),
!C     X             APT(2,2,2),AC(2,2),BC(2,2),AFP,E(2,2),DELT,AREA,
!C     X             AB,BB,BT(2),XNC(3),UH(3),P,AMR,FM,T4(3),ALIM(2,2)
!C
!C
!C      As found in PLELP
!C      COMMON/TEMPVS/DMNT(3,3),TEMP(3,3),B(3,3),XMN(3),RLN(3),XMM(3),
!C     *               TM(3),R(3),RM(3),DMNWN(3),RLM(3),RN(3),VMN(3),VR(3),
!C     *               WNM(3),WCM(3),WCN(3),VREL(3),FFM(3),FR(3),TQM(3),
!C     *               TQN(3),TQNT(3),T(3),H(3),TH(3),XH(3),RMD(3),RND(3),
!C     *               TD(3),TT4(3,4),TT5(3,4),XNC(3),UH(3),P,AMR,FM,CF,
!C     *               VRM,VRT,VRTS,VRTEST,TF,ELOSS,MCF,NCF
!C
!C     As found in PLSEGF; is shared by PLELP, PLSEGF and SEGSEG.
!C      DIMENSION     DMNT(3,3),TEMP(3,3),B(3,3),XMN(3),RLN(3),XMM(3),
!C     *               TM(3),R(3),RM(3),DMNWN(3),RLM(3),RN(3),VMN(3),VR(3),
!C     *               WMN(3),WCM(3),WCN(3),VREL(3),FFM(3),FR(3),TQM(3),
!C     *               TQN(3),TQNT(3),T(3),H(3),T1(3),T2(3),RMD(3),RND(3)
!C     *               TD(3),TT4(3,4),TT5(3,4),T3(3),T4(3),P,AMR,FM,CF,
!C     *               VRM,VRT,VRTS,VRTEST,TF,ELOSS,MCF,NCF,T5(3),T6(3)
!C     As found in SEGSEG
!C      COMMON/TEMPVS/DMNT(3,3),TEMP(3,3),B(3,3),XMN(3),RLN(3),XMM(3),
!C     *               TM(3),R(3),RM(3),DMNWN(3),RLM(3),RN(3),VMN(3),VR(3),
!C     *               WNM(3),WCM(3),WCN(3),VREL(3),FFM(3),FR(3),TQM(3),
!C     *               TQN(3),TQNT(3),T(3),H(3),T1(3),T2(3),RMD(3),RND(3),
!C     *               TD(3),TT4(3,4),TT5(3,4),T3(3),T4(3),P,AMR,FM,CF,
!C     *               VRM,VRT,VRTS,VRTEST,TF,ELOSS,MCF,NCF,T5(3),T6(3)
!C
!C     As found in HYEST.
```

```
!C      COMMON/TEMPVS/D12(3,3),A(3,3),B(3,3),XMN(3),RLN(3),XMM(3),
!C     *              T(3),R(3),C(3,3),V(7)
!C
!C     As found in HYLPX.
!C      COMMON/TEMPVS/D12(3,3),P(3,3),Q(3,3),XMN(3),RLN(3),XMM(3),
!C     *              R(3),H(3),D(3,3),V(7)
!C
!C     As found in HYNTR.
!C      COMMON/TEMPVS/D12(3,3),A(3,3),B(3,3),XMN(3),RLN(3),XMM(3),
!C     *              AZ(3),R(3)
!C
!C
!C*****************************************************************************
!C
!C    /VIN_TEMPVS/
!C
!C    The first 2 lines of /TEMPVS/ were shared by VINPUT, VINO12,
!C     VINO34, VSPLIN and VINTST.
!C
      REAL   ( KIND = IREAL_HIGH )
     &                  ANGLE, ATAB, AX, DVEH, VMEG, VMEGD,
     &                  X0, XACOMP, XDOT0
      DIMENSION         ANGLE(3), ATAB(15,MAXVT3), AX(3), DVEH(3,3),
     &                  VMEG(3), VMEGD(3), X0(3), XACOMP(3),
     &                  XDOT0(3)
!C
!C    COMMON/TEMPVS/ X0(3),XDOT0(3),XACOMP(3),AX(3),ANGLE(3),
!C    *               ATAB(15,MAXVT3),DVEH(3,3),VMEG(3),VMEGD(3)
!C
!C
!C*****************************************************************************
!C
!C    New global variables that had been local variables in Subroutine
!C     INPUT_VEHICLE.  They are the linear and angular initial
!C     parameters for segments that have been specified to be vehicles.
!C
      REAL   ( KIND = IREAL_HIGH )
     &              RINIT_SEGLP, RINIT_SEGLV, RINIT_SEGLA,
     &              RINIT_D, RINIT_WMEG, RINIT_WMEGD
      DIMENSION  RINIT_D(3,3,MAXSEG), RINIT_SEGLA(3,MAXSEG),
     &           RINIT_SEGLP(3,MAXSEG), RINIT_SEGLV(3,MAXSEG),
     &           RINIT_WMEG(3,MAXSEG), RINIT_WMEGD(3,MAXSEG)

!C
!C*****************************************************************************
!C
      END MODULE MODULE_STANDARD
```

A.2 MODULE_FLEXIBLE Source Code

```
      MODULE  MODULE_FLEXIBLE
!C
!C                                                    Rev. V.2 06/01/2000
!C
!C
!C    This module contains the COMMON BLOCKs related to the flexible
!C     segment option.
!C
      USE  MODULE_STANDARD,  ONLY:
     &       INTEGER_STD, IREAL_HIGH, MAXSEG, MAXJNT, MAXDEF,
     &                                MXMOD, MXNOD
!C
      IMPLICIT  NONE
!C
!C
!C*******
!C
!C    /FXBODY/
!C
      REAL  ( KIND = IREAL_HIGH )  QNOD, WNOD, FMODES, FIK
      DIMENSION  QNOD(3,MXNOD,3*MAXDEF), WNOD(MXNOD,MAXDEF),
     &           FMODES(6*MXNOD,MXMOD,3*MAXDEF), FIK(3,MXNOD,MAXDEF)
!C
!C      COMMON /FXBODY/  QNOD, WNOD, FMODES, FIK                          DEFJNT
!C
!C
!C*******
!C
!C    /FXCOEF/
!C
      REAL  ( KIND = IREAL_HIGH )
     &                  WPI, PHPI, U1P, A21P, U2P, UAP, A11P, A12P,
     &                  A22P, AA1P, AA2P, B1A, B2A, A11F, A22F
      DIMENSION   WPI(3,3,MAXDEF), PHPI(3,3,MAXDEF), U1P(3,MAXDEF),     DEFORM
     &            A21P(3,3,2*MAXJNT), U2P(3,MAXDEF), UAP(MXMOD,MAXDEF), DEFORM
     &            A11P(3,3,2*MAXJNT), A12P(3,3,2*MAXJNT),               DEFORM
     &            A22P(3,3,2*MAXJNT), AA1P(MXMOD,3,2*MAXJNT),           DEFORM
     &            AA2P(MXMOD,3,2*MAXJNT), B1A(3,MXMOD,2*MAXJNT),        DEFORM
     &            B2A(3,MXMOD,2*MAXJNT), A11F(3,3,2*MAXJNT),            DEFORM
     &            A22F(3,3,2*MAXJNT)                                    DEFORM
!C
!C      COMMON /FXCOEF/  WPI, PHPI, U1P, A21P, U2P, UAP, A11P, A12P,
!C     *                 A22P, AA1P, AA2P, B1A, B2A, A11F, A22F
!C
!C
!C********
!C
!C     /FXFRC/
!C
      INTEGER  ( KIND = INTEGER_STD )  NODSD, NODFR
      DIMENSION  NODSD(2,20), NODFR(5)
!C
      REAL  ( KIND = IREAL_HIGH )  PURTQ, YFB
      DIMENSION  PURTQ(3,MAXDEF), YFB(3,MAXDEF)
!C
!C    COMMON /FXFRC /  PURTQ, YFB, NODSD, NODFR
!C
!C********
!C
!C     /FXINT/
!C
      INTEGER  ( KIND = INTEGER_STD )  NEQP
!C
!C      COMMON /FXINT /  NEQP
!C
```

```
!C********
!C
!C     /FXJROT/
!C
      INTEGER   ( KIND = INTEGER_STD )  NTDEF, IDSEG, IDJNT, JROUT
      DIMENSION  IDSEG(MAXDEF), IDJNT(2,MAXDEF), JROUT(MAXDEF)
!C
      REAL   ( KIND = IREAL_HIGH )  ROTJ, ROTVJ, DNP, ANF, CN
      DIMENSION  ROTJ(3,2*MAXJNT), ROTVJ(3,2*MAXJNT),
     &           DNP(3,3,2*MAXJNT), ANF(3,3,2*MAXJNT),                    DEFJNT
     &           CN(3,2*MAXJNT)
!C
!C     COMMON /FXJROT/  ROTJ, ROTVJ, DNP, ANF, CN, NTDEF, IDSEG, IDJNT
!C
!C*******
!C
!C     /FXNVEL/
!C
      REAL   ( KIND = IREAL_HIGH )   ASAD,  WNP
      DIMENSION  ASAD(3,2*MAXJNT), WNP(3,2*MAXJNT)
!C
!C      COMMON /FXNVEL/  ASAD, WNP
!C
!C*******
!C
!C     /FXOUT/
!C
      INTEGER   ( KIND = INTEGER_STD )   NFBPR, NODPR
      DIMENSION  NFBPR(20,3), NODPR(20,3)
!C
      REAL   ( KIND = IREAL_HIGH )   T91, T92
      DIMENSION  T91(3), T92(3)
!C
!C      COMMON /FXOUT /  T91, T92, NFBPR, NODPR
!C
!C*******
!C
!C     /FXSING/
!C
      REAL   ( KIND = IREAL_HIGH )   TAM, RAM
      DIMENSION  TAM(3,MXMOD,MAXDEF), RAM(3,MXMOD,MAXDEF)
!C
!C     COMMON /FXSING/  TAM, RAM
!C
!C********
!C
!C     /FXVAR/
!C
      INTEGER   ( KIND = INTEGER_STD )  NFBOD, IBODN, NNOD, NMOD, NODJ
      DIMENSION  IBODN(MAXDEF), NNOD(MAXDEF), NMOD(3*MAXDEF),
     &           NODJ(3,2,MAXSEG)
!C
      REAL   ( KIND =IREAL_HIGH )  RSTF, RDMP, TTM, SAIM, AMP, AMV, AMA
      DIMENSION  RSTF(MXMOD,MAXDEF), RDMP(MXMOD,MAXDEF), TTM(MAXDEF),
     &           SAIM(3,MXMOD,MAXDEF), AMP(MXMOD,3*MAXDEF),
     &           AMV(MXMOD,3*MAXDEF), AMA(MXMOD,3*MAXDEF)
!C
!C     COMMON /FXVAR/  RSTF, RDMP, TTM, SAIM, AMP, AMV, AMA, NFBOD,      DEFJNT
!C    *                IBODN, NNOD, NMOD, NODJ                           DEFJNT
!C
!C********
!C
!C     /FXXTRA/
!C
      REAL   ( KIND = IREAL_HIGH )  HB0, HT0, DBN, FMODM
      DIMENSION  HB0(3,2*MAXJNT), HT0(3,3,2*MAXJNT),
     &           DBN(3,3,2*MAXJNT), FMODM(3,MXMOD,2*MAXJNT)
!C
!C     COMMON /FXXTRA/  HB0, HT0, DBN, FMODM
!C
!C
```

```
!C********
!C
!C       /OLDDAT/
!C
      REAL   ( KIND = IREAL_HIGH )    AMPOLD, FMODO, ROTOLD
      DIMENSION          AMPOLD(MXMOD,MAXDEF), FMODO(3,MXMOD,2*MAXJNT),
     &                   ROTOLD(3,2*MAXJNT)
!C
!c       COMMON /OLDDAT/ AMPOLD(MXMOD,MAXDEF), FMODO(3,MXMOD,2*MAXJNT),
!c      &                ROTOLD(3,2*MAXJNT)
!C
!C
!C*********************
!C*********************
!C
!C    Parameters associated with the flexible/deformable segments.
!C
      REAL   ( KIND = IREAL_HIGH )   SMALL_START
      PARAMETER ( SMALL_START = 1.0E12_IREAL_HIGH )
!C
      END MODULE  MODULE_FLEXIBLE
```

A.3 MODULE_WATER Source Code

```
      MODULE  MODULE_WATER
!C
!C                                                    Rev. V.2 06/01/2000
!C
!C    This MODULE contains the water force related COMMON BLOCKs.
!C
      USE  MODULE_STANDARD,  ONLY:
     &        INTEGER_STD, IREAL_HIGH, MAXELP, MAXSEG   ! parameters
!C
      IMPLICIT  NONE
!C
!C    SAVE all variables, since MODULE_WATER is not referenced by
!C     .MAIN, making it possible for these variables to become
!C     undefined during the execution of the program, when no
!C     subroutines are being executed that reference this module.
!C
      SAVE
!C
!C********
!C
!C     /ELPDAT/
!C
      INTEGER  ( KIND = INTEGER_STD )  NELPS
!C
      REAL  ( KIND = IREAL_HIGH )  DELP
      DIMENSION  DELP(3,3,MAXELP)
!C
!C      COMMON /ELPDAT/  DELP, NELPS
!C
!C********
!C
!C     /WATGRD/
!C
      REAL  ( KIND = IREAL_HIGH )  RPH, RNX
!C
!C     COMMON /WATGRD/  RPH, RNX
!C
!C********
!C
!C    /WATINF1/
!C
      INTEGER  ( KIND = INTEGER_STD )
     &         MOUTHS, MOUTHE, NEBODY, NSBODY, NWATER, NPFD, NEW,
     &         NWSE, NELPFD, KPFD
      DIMENSION  NWSE(2,MAXELP), NELPFD(5), KPFD(25)
!C
      REAL  ( KIND = IREAL_HIGH )
     &           BDPFD, PFDWT, DMOUTH, COED, TBDV, COEL, CADDM
      DIMENSION  BDPFD(30,25), PFDWT(5,5), DMOUTH(3),
     &           COED(MAXELP+25), COEL(MAXELP+25), CADDM(6,MAXELP+25)
!C
!C      COMMON /WATINF1/  BDPFD, PFDWT, DMOUTH, COED, TBDV, COEL,
!C     *                  CADDM, MOUTHS, MOUTHE, NEBODY, NSBODY,
!C     *                  NWATER, NPFD, NEW, NWSE, NELPFD, KPFD
!C
!C********
!C
!C     /WATINF2/
!C
      INTEGER  ( KIND = INTEGER_STD )  NPE
!C
      REAL  ( KIND = IREAL_HIGH )   DPFD
      DIMENSION  DPFD(3,3,25)
!C
!C      COMMON /WATINF2/  DPFD, NPE
!C
```

```
!C********
!C
!C     /WAVEDAT/
!C
      INTEGER   ( KIND = INTEGER_STD )  ISPD, NWAVES
!C
      REAL   ( KIND = IREAL_HIGH )
     &                  WOFSET, WFRAME, WD, WDEP, WNUM, WAMP, WDIR,
     &                  WPHS, FREQ, SWKH, WGAM, WSPD
      DIMENSION  WOFSET(3), WFRAME(3), WD(3,3), WNUM(10), WAMP(10),
     &           WDIR(10), WPHS(10), FREQ(10), SWKH(10)
!C
!C      COMMON /WAVEDAT/ WOFSET, WFRAME, WD, WDEP, WNUM, WAMP, WDIR,
!C     *                  WPHS, FREQ, SWKH, WGAM, WSPD, ISPD, NWAVES
!C
!C********
!C
!C     /WFACOP/
!C
      INTEGER   ( KIND = INTEGER_STD )  NSEQN, NELL, NELOUT, ITYPE
      DIMENSION  NELL(5), NELOUT(5,2,MAXELP+25), ITYPE(6)
!C
      REAL   ( KIND = IREAL_HIGH )  BRI, TENY, WAREA, DISTM, ALFA1, ALFA2
!C
!C      COMMON /WFACOP/  BRI, TENY, WAREA, DISTM, ALFA1, ALFA2, NSEQN,
!C     *                  NELL, NELOUT, ITYPE
!C
!C********
!C
!C     /WMASS/
!C             This COMMON BLOCK was in Subroutine ADDMAS and DAUX
!C             but not in BLOCK_DATA of version V.1.
!C
      REAL   ( KIND = IREAL_HIGH )  WX, RWX, WXX, RWXX
      DIMENSION  WX(3,MAXSEG), RWX(3,MAXSEG), WXX(MAXSEG), RWXX(MAXSEG)
!C
!C      common /wmass/   wx(3,MAXSEG),rwx(3,MAXSEG),wxx(MAXSEG),
!C     *                 rwxx(MAXSEG)
!C
!C*********
!C
!C     /WRESLTS/
!C
      REAL   ( KIND = IREAL_HIGH )  BUOY, WEXF, ADDM, DRAG, BVL, AREA
      DIMENSION   BUOY(6,MAXELP+25), WEXF(6,MAXELP+25),
     &            ADDM(3,MAXELP+25), DRAG(6,MAXELP+25),
     &            BVL(MAXELP+25), AREA(MAXELP+25)
!C
!C      COMMON /WRESLTS/  BUOY, WEXF, ADDM, DRAG, BVL, AREA
!C
!C
!C*********
!C
!C    This is a temporary module, which contains the /TEMPFD/
!C     variables used by the water subroutines ADDMAS, BOYCTR,
!C     and WATINP.
!C
!C    Was  /TEMPFD/
!C
      INTEGER   ( KIND = INTEGER_STD )  KSEG_WATER, KELL, KELT
!C
      REAL   ( KIND = IREAL_HIGH )
     &                  T2, T5, P1, CENTW, FL, F1, FF_WATER, WDSE,
     &                  TSN, TN, BET, BTE, E11, E12, E22, C1, S1,
     &                  UU_WATER, VV
      DIMENSION  T2(3), T5(3), P1(3), CENTW(3), FL(100,6), F1(100,6),
     &           FF_WATER(6), WDSE(3,3,MAXELP+25), TSN(3), TN(3),
     &           UU_WATER(3), VV(3)
!C
!C
      END MODULE  MODULE
```

Appendix B – Listings of Subroutine

THIS PAGE LEFT BLANK INTENTIONALLY

Appendix B.1: Migration of V.1 Subroutines to V.3 Subroutines

(Subroutines that are new to V.3 or have been renamed in V.3 are bolded)

V.1	V.3
	ACTUATOR_TORQUE (was USER)
ADDMAS	**renamed to WATER_ADDED_MASS**
ADJUST	ADJUST
AIRBAG	AIRBAG
AIRBG1	**renamed to INPUT_AIRBAGS**
AIRBG3	AIRBG3
AIRBGG	AIRBGG
APPLY	APPLY
BELTG	BELTG
BELTRT	BELTRT
BGG	BGG
BINPUT	**deleted**
BLKDTA	BLKDTA
BLOCKDTA	**deleted**
BOYCTR	BOYCTR
CFACTT	CFACTT
CHAIN	CHAIN
	CHECK_COMMENT
	CHECK_HIGH_VALUE
	CHECK_ROT_SEQUENCE
	CHECK_ROTATION_ORDER
CHKREF	CHKREF
CINPUT	**renamed to INPUT_FUNCTIONS**
CMPUTE	CMPUTE
COMPSYS	**deleted**
CONTCT	CONTCT
	CONVERT_TIME
CROSS	CROSS
CUTARE	CUTARE
	DATE_TIME
DAUX	DAUX
DAUX11	DAUX11
DAUX12	DAUX12
DAUX22	DAUX22
	DAUX22_SUB
DAUX31	DAUX31
DAUX32	DAUX32
DAUX33	DAUX33
DAUX44	DAUX44
DAUX55	DAUX55
	DAUX_SETUP
	DEF_NEW_CUBIC
	DEF_NEW_QUADRATIC
DHHPIN	DHHPIN
DINTG	DINTG
	DINTG_BACKUP
	DINTG_HALF
DOT31	DOT31
DOT33	DOT33
DOTT31	DOTT31
DOTT33	DOTT33
DOTVEC	DOTVEC
DRCIJK	DRCIJK
DRCQUA	DRCQUA
DRCYPR	DRCYPR
DRGCHK	**renamed to WATER_DRAG_CHK**
DRIFT	DRIFT
DSETD	DSETD
DSETQ	DSETQ
DSMSOL	DSMSOL
DZP	DZP

V.1	V.3
EDEPTH	EDEPTH
EFUNCT	EFUNCT
EJOINT	EJOINT
	EJOINT_TORQUE
ELARE3	ELARE3
ELAREA	**deleted**
ELONG	ELONG
ELTIME	**E_ELTIME (renamed)**
EQUILB	EQUILB
	EQUILB_SOLVE
	EQUILB_SOLVE_SUB
ETA	**renamed to WAVE_HEIGHT**
EULRAD	EULRAD
EVALFD	EVALFD
	EVALFD_INTEGRAL
	EVALFD_POLY
	EVALFD_TABLE
FDINIT	FDINIT
FDUX11	FDUX11
FDUX12	FDUX12
FDUX22	FDUX22
FILES	**renamed to INPUT_FILES**
FINPUT	**deleted**
FJNTVC	FJNTVC
FLXSEG	FLXSEG
FNAME	FNAME
FNTERP	FNTERP
	FRAME_WINDOW
	FORCE_TORQUE
FRCDFL	FRCDFL
FSETUP	FSETUP
FSMSOL	FSMSOL
	FSMSOL_STOP
FSPDMP	FSPDMP
	FUNC_RATE_DEP
FUNCHK	FUNCHK
FXCAHW	FXCAHW
FXCBLT	FXCBLT
FXCPLS	FXCPLS
FXINPT	**renamed to INPUT_DEFORM**
FXMACC	FXMACC
FXMASS	FXMASS
FXMODD	FXMODD
FXMODV	FXMODV
FXMODV0	FXMODV0
	GET_MONTH_NAME
GLOBAL	GLOBAL
HBELT	HBELT
HBPLAY	HBPLAY
	HBPLAY_POINTS
HEDING	HEDING
	HEDING_ACTUATORS
	HEDING_ANG_DISPL
	HEDING_BODY_PROP
	HEDING_FORCES
	HEDING_HCARDS
	HEDING_JNT_PARM
	HEDING_JOINT_FORCES
	HEDING_WATER
	HEDING_WIND
HERRON	HERRON

V.1	V.3
HICCSI	HICCSI
HINPUT	**renamed to INPUT_HARNESS**
	HPOINT_DROP
HPTURB	HPTURB
	HPTURB_SETUP
HSETC	HSETC
	HSETC_SUB
HYAFB	HYAFB
HYBND	HYBND
HYBOX	HYBOX
HYDAD	HYDAD
HYEST	HYEST
HYFCN	HYFCN
HYLIM	HYLIM
HYLPR	HYLPR
	HYLPR_PIVOT
HYLPX	HYLPX
HYNTR	HYNTR
HYPEN	HYPEN
HYREA	HYREA
HYSOL	HYSOL
HYVAL	HYVAL
HYVBX	HYVBX
HYVFN	HYVFN
IMPLS2	IMPLS2
IMPULS	IMPULS
INTIAL	**renamed to INPUT_INITIAL_CONDITIONS**
	INITIALIZE
	INPUT_ACARDS
	INPUT_AIRBAG_FORCE
	INPUT_AIRBAGS (was AIRBG1)
	INPUT_BCARDS
	INPUT_BELT_FORCE
	INPUT_BELTS
	INPUT_CONSTRAINTS
	INPUT_DCARDS
	INPUT_DEFORM (was FXINPT)
	INPUT_ELLIPSOIDS
	INPUT_EQUILB
	INPUT_FCARDS
	INPUT_FILES (was FILES)
	INPUT_FLEX
	INPUT_FORCE_TORQUE
	INPUT_FUNCTIONS (was CINPUT)
	INPUT_GLOBALGRAPHIC_FORCE
	INPUT_H10_CARDS
	INPUT_H11_CARDS
	INPUT_H1_H3_CARDS
	INPUT_H4_H9_CARDS
	INPUT_H7_CARDS
	INPUT_HARNESS (was HINPUT)
	INPUT_HCARDS
	INPUT_INITIAL_CONDITIONS (was INTIAL)
	INPUT_JOINT_TORQUE
	INPUT_JOINTS
	INPUT_LINEAR (was INTLIN)
	INPUT_ORIENT (was INTANG)
	INPUT_PER_FLOAT_DEV
	INPUT_PLANE_FORCE
	INPUT_PLANES
	INPUT_PROJANG (was INPROJ)
	INPUT_ROBOTICS (was ROBINP)
	INPUT_SEG_SEG_FORCE
	INPUT_SPRING_DAMPERS
	INPUT_SYMMETRY
	INPUT_VEHICLE (was VINPUT)
	INPUT_WATER (was WATINP)
	INPUT_WATER_ELLIPSOIDS

V.1	V.3
	INPUT_WATER_OUTPUT
	INPUT_WAVES
	INPUT_WIND
	INPUT_WIND_FORCE
INPROJ	**renamed to INPUT_PROJANG**
INRTIA	INRTIA
INTANG	**renamed to INPUT_ORIENT**
INTERS	INTERS
	INTERS_SOLVE
INTLIN	**renamed to INPUT_LINEAR**
INVERS	INVERS
JNTFNC	JNTFNC
JNTROT	JNTROT
KINPUT	**deleted**
LTIME	**L_LTIME (renamed)**
LUDCMP	LUDCMP
	LUNUM_FORCES
	LUNUM_HCARDS
	LUNUM_WATER (was WATSET)
LUPIVT	LUPIVT
LUSUB	LUSUB
MAIN_ATB	MAIN_ATB
MAT31	MAT31
MAT33	MAT33
	MODULE_FLEXIBLE
	MODULE_STANDARD
	MODULE_WATER
MULPLY	MULPLY
NUMCHR	NUMCHR
ORTHO	ORTHO
OUTPUT	OUTPUT
	OUTPUT_BODY_PROP
	OUTPUT_FORCES
	OUTPUT_H9_CARDS
	OUTPUT_HCARDS
	OUTPUT_JOINTS
	OUTPUT_LUNUM
	OUTPUT_SETUP
	OUTPUT_WATER (was WATOUT)
PANEL	PANEL
PDAUX	PDAUX
PFDBOY	PFDBOY
PFDFRC	PFDFRC
PFDWXC	PFDWXC
PLEDG	PLEDG
PLELP	PLELP
PLREA	PLREA
PLSEGF	PLSEGF
POSTPR	POSTPR
PRINT	PRINT
PRNCIPAL	PRNCIPAL
QSET	QSET
QUAT	QUAT
RCRT	RCRT
	READ_TAPE_8
ROBINP	**renamed to INPUT_ROBOTICS**
ROT	ROT
ROTATE	ROTATE

V.1	V.3
SEGSEG	SEGSEG
SETUP1	SETUP1
SETUP2	SETUP2
SIMPSN	SIMPSN
SINPUT	**deleted**
SOLVA	SOLVA
SOLVR	SOLVR
SPDAMP	SPDAMP
SPLINE	SPLINE
SPRNGF	**deleted**
TILDE	TILDE
TRIGFS	TRIGFS
TRNPOS	TRNPOS
U1ASC	U1ASC
U1ASCD	U1ASCD
U1ASCH	U1ASCH
U1OLD	U1OLD
UNIT1	UNIT1
UNTVEC	UNTVEC
UPDATE	UPDATE
	UPDATE_EULER_JOINTS
	UPDATE_FRC_DEF_CURVE (was UPDFDC)
	UPDATE_JOINTS
	UPDATE_PFD (was UPDPFD)
	UPDATE_TAB
UPDFDC	**renamed to UPDATE_FRC_DEF_CURVE**
UPDPFD	**renamed to UPDATE_PFD**
USER	**renamed to ACTUATOR_TORQUE**
VECANG	**deleted**
VECMAG	VECMAG
VEHPOS	VEHPOS
VINITL	VINITL
VINO12	VINO12
VINO34	VINO34
VINPUT	**renamed to INPUT_VEHICLE**
VINTST	VINTST
VISCOS	VISCOS
VISPR	VISPR
	VISPR_TORQUE
VPATH	VPATH
VPATH2	VPATH2
VSPLIN	VSPLIN
	WATER_ADDED_MASS (was ADDMAS)
	WATER_DRAG_CHK (was DRGCHK)
	WATER_ELLIP_FORCE (was WELFOR)
	WATER_FORCE (was WFORCE)
	WATER_PNT_VELOCITY (was WAVEL)
	WAVE_HEIGHT (was ETA)
WATHED	**renamed to HEDING_WATER**
WATINP	**renamed to INPUT_WATER**
WATOUT	**renamed to OUTPUT_WATER**
WATSET	**renamed to LUNUM_WATER**
WAVEL	**renamed to WATER_PNT_VELOCITY**
WBAREA	WBAREA
WELFOR	**renamed to WATER_ELLIP_FORCE**
WEXPHI	WEXPHI
WFORCE	**renamed to WATER_FORCE**
	WIND_AREA
	WIND_GRID
WINDY	WINDY
XDY	XDY
YPRDEG	YPRDEG

Appendix B.2: V.3 Subroutines by Function

Airbags
- AIRBAG
- AIRBG3
- AIRBGG
- BGG
- PANEL
- RCRT

Contact forces
- CONTCT
- FXCAHW (*1)

External applied force torque
- FORCE_TORQUE

Function setup evaluation
- DEF_NEW_CUBIC
- DEF_NEW_QUADRATIC
- EVALFD
- EVALFD_INTEGRAL
- EVALFD_POLY
- EVALFD_TABLE
- FDINIT
- FRCDFL
- FUNC_RATE_DEP
- FUNCHK
- UPDATE_FRC_DEF_CURVE
- UPDATE_TAB

Harness_belts
- HBELT
- HBPLAY
- HBPLAY_POINTS
- HPOINT_DROP
- HPTURB
- HPTURB_SETUP
- HSETC
- HSETC_SUB

Hyper_ellipsoids
- HYABF
- HYBND
- HYBOX
- HYDAD
- HYEST
- HYFCN
- HYLIM
- HYLPR
- HYLPR_PIVOT
- HYLPX
- HYNTR
- HYPEN
- HYREA
- HYSOL
- HYVAL
- HYVBX
- HYVFN

Initialize_subroutines
- BLKDTA
- EQUILB
- EQUILB_SOLVE
- EQUILB_SOLVE_SUB
- INITIALIZE
- ROTATE

Input_subroutines
- CHECK_COMMENT
- CHECK_ROT_SEQUENCE
- CHECK_ROTATION_ORDER
- COMPUTE_INIT_ANG_VEL
- INPUT_ACARDS
- INPUT_AIRBAG_FORCE
- INPUT_AIRBAGS
- INPUT_BCARDS
- INPUT_BELT_FORCE
- INPUT_BELTS
- INPUT_CONSTRAINTS
- INPUT_DCARDS
- INPUT_DEFORM
- INPUT_ELLIPSOIDS
- INPUT_EQUILB
- INPUT_FCARDS
- INPUT_FILES
- INPUT_FLEX
- INPUT_FORCE_TORQUE
- INPUT_FUNCTIONS
- INPUT_GLOBALGRAPHIC_FORCE
- INPUT_H1_H3_CARDS
- INPUT_H10_CARDS
- INPUT_H11_CARDS
- INPUT_H4_H9_CARDS
- INPUT_H7_CARDS
- INPUT_HARNESS
- INPUT_HCARDS
- INPUT_INITIAL_CONDITIONS
- INPUT_JOINT_TORQUE
- INPUT_JOINTS
- INPUT_LINEAR
- INPUT_ORIENT
- INPUT_PER_FLOAT_DEV
- INPUT_PLANE_FORCE
- INPUT_PLANES
- INPUT_PROJANG
- INPUT_ROBOTICS
- INPUT_SEG_SEG_FORCE
- INPUT_SPRING_DAMPERS
- INPUT_SYMMETRY
- INPUT_VEHICLE
- INPUT_WATER
- INPUT_WATER_ELLIPSOIDS
- INPUT_WATER_OUTPUT
- INPUT_WAVES
- INPUT_WIND
- INPUT_WIND_FORCE

Integration_subroutines
- ADJUST
- CMPUTE
- DINTG
- DINTG_BACKUP
- DINTG_HALF
- DZP
- IMPLS2
- IMPULS
- INTEGRATE_TIME
- QSET
- TRIGFS
- UPDATE
- UPDATE_CONSTRAINTS

Joints
- ACTUATOR_TORQUE (*2)
- APPLY
- DHHPIN
- DRCIJK
- DRIFT
- EFUNCT
- EJOINT
- EJOINT_TORQUE
- EULRAD
- FJNTVC (*1)
- FNTERP
- GLOBAL
- HERRON
- JNTFNC (*2)
- JNTROT (*1)
- UPDATE_EULER_JNTS
- UPDATE_JOINTS
- VISCOS
- VISPR
- VISPR_TORQUE

Main program / modules
- MAIN_ATB
- MODULE_FLEXIBLE
- MODULE_STANDARD
- MODULE_WATER

Math_utilities
- CFACTT
- CROSS
- DOT31
- DOT33
- DOTT31
- DOTT33
- DOTVEC
- DRCQUA
- DRCYPR
- DSETD
- DSETQ
- DSMSOL
- FSMSOL
- FSMSOL_STOP
- INVERS
- LUDCMP
- LUPIVT
- LUSUB
- MAT31
- MAT33
- MULPLY
- ORTHO
- QUAT
- ROT
- SIMPSN
- SPLINE
- TILDE
- TRNPOS
- UNTVEC
- VECMAG
- XDY
- YPRDEG

Output_subroutines
- CHECK_HIGH_VALUE
- CHKREF
- FNAME
- HEDING
- HEDING_ACTUATORS
- HEDING_ANG_DISPL
- HEDING_BODY_PROP
- HEDING_FORCES
- HEDING_HCARDS
- HEDING_JNT_PARM
- HEDING_JOINT_FORCES
- HEDING_WATER
- HEDING_WIND
- HICCSI
- INRTIA
- LUNUM_FORCES
- LUNUM_HCARDS
- LUNUM_WATER
- NUMCHR
- OUTPUT
- OUTPUT_BODY_PROP
- OUTPUT_FORCES
- OUTPUT_H9_CARDS
- OUTPUT_HCARDS
- OUTPUT_JOINTS
- OUTPUT_LUNUM
- OUTPUT_SETUP
- OUTPUT_WATER
- POSTPR
- PRINT
- PRNCIPAL
- READ_TAPE_8
- U1ASC
- U1ASCD
- U1ASCH
- U1OLD
- UNIT1

Plane/seg_seg/seg_forces
- EDEPTH
- FXCPLS (*1)
- INTERS
- INTERS_SOLVE
- PLEDG
- PLELP
- PLREA
- PLSEGF
- SEGSEG

Simple_belt_forces
- BELTG
- BELTRT
- ELONG
- FXCBLT (*1)

Solver_subroutines
- CHAIN
- DAUX
- DAUX_SETUP
- DAUX11
- DAUX12
- DAUX22
- DAUX22_SUB
- DAUX31
- DAUX32
- DAUX33
- DAUX44
- DAUX55
- FDUX11 (*1)
- FDUX12 (*1)
- FDUX22 (*1)
- FLXSEG
- FSETUP (*1)
- FXMACC (*1)
- FXMASS (*1)
- FXMODD (*1)
- FXMODV (*1)
- FXMODV0 (*1)
- PDAUX
- SETUP1
- SETUP2

Spring_dampers
- FSDMP
- SPDAMP

System_utilities
- CONVERT_TIME
- DATE_TIME
- E_ELTIME
- FRAME_WINDOW (*3)
- GET_MONTH_NAME
- L_LTIME

Vehicles
- VEHPOS
- VINITL
- VINO12
- VINO34
- VINTST
- VPATH
- VPATH2
- VSPLIN

Water_forces
- BOYCTR
- CUTARE
- ELARE3
- PFDBOY
- PFDFRC
- PFDWXC
- UPDATE_PFD
- WATER_ADDED_MASS
- WATER_DRAG_CHK
- WATER_ELLIP_FORCE
- WATER_FORCE
- WATER_PNT_VELOCITY
- WAVE_HEIGHT
- WBAREA
- WEXPHI

Wind_forces
- SOLVA
- SOLVR
- WIND_AREA
- WIND_GRID
- WINDY

Notes
- *1: Non I/O subroutines used only for deformable body option
- *2: Non I/O subroutines used only for joint actuator option
- *3: System subroutine used for the Compaq QuickWin window setup

THIS PAGE LEFT BLANK INTENTIONALLY

Appendix C – Miscellaneous Coding Change Notes

THIS PAGE LEFT BLANK INTENTIONALLY

APPENDIX C – Miscellaneous Coding Change Notes

This Appendix contains selected notes for some of the modifications made to Version V.1 of the code. These notes are not meant to be all-inclusive and do not contain notes of all changes. Please use these notes, the main body of the report, and the comments provided in the code to obtain a complete understanding of all the changes made to the code. The most significant changes were to place all of the COMMON BLOCKs from BLOCK_DATA into several modules, then comment out the COMMON BLOCKs, so that the module only typed and dimensioned the variables. All calls to these COMMON BLOCKs in all the subroutines and the associated PARAMETER statements were deleted and replaced with USE-ONLY statements. All local variables were explicitly typed and dimensioned. All DO loops that terminated on a executable statement were replaced either with an END DO, if the DO loop was relatively short, or a numbered CONTINUE statement if the DO loop was long. All nested DO loops that ended on the same statement were replaced with unique termination statements. DO loop indentation was also added in some places. A few COMMENT statements were added, mostly blanks, and the "&" was used to replace the * in the continuation lines. Furthermore, !'s were placed in front of the C's in many instances to head towards f95 compliance. In rare instances, the names of local variables were changed if there existed a variable in one of the modules that had the same name. This was done to avoid any confusion between local and global variables, as well as to avoid any potential errors that might occur if a module is used with a USE statement without the ONLY option. All subroutines were compiled on the DEC (now Compaq) Visual Fortran 6.0.A compiler. Two compliance checks were made. On the first pass, f90 compatibility was turned on. F95 compatibility was turned on for the second pass.

Modules

MODULE_STANDARD: – created from COMMON BLOCKs for the most part associated with the standard ATB code. COMMON BLOCKs were commented out, all variables were explicitly typed and dimensioned. Defined the single-dimensioned arrays XTEST_SNGL and REGT_SNGL for use by Subroutine DINTG. Used the /TEMPVS/ from DAUX as the definition of /TEMPVS/. Also added variables MS_SEG, NF_FUNCT, KTITLE and JTITLE from the /TEMPVS/ that was shared by Subroutines CINPUT, FDINIT, FINPUT, and HINPUT. Added the /TEMPVS/ that is shared by the harness belt subroutines, and renamed most of the variables in this common block [see the code for a listing of all the renamed variables]. Defined BLOSS and HLOSS as equivalenced with variables from the harness /TEMPVS/. Added /COUT/ from Subroutine HEDING, though it was not included in BLOCK DATA. Also added the /TEMPVS/ that is shared by Subroutines POSTPR, HEDING, HEDINGX, with the variables as found in Subroutine HEDING. Used the new form of the CHARACTER assignment for CHARACTERs CGS, JS, and OUTFIL. Added common blocks /BAGDIM/, /COUTN/ and /COUTFMT/ from Subroutine U1ASCD, as well as the associated Parameters. Added /AIRBAG_TEMPVS/ created from the /TEMPVS/ that was shared by Subroutines AIRBAG and AIRBGG. The COMMON BLOCK was commented out, all variables were explicitly typed and dimensioned. Added /BELT_TEMPVS/ that was created from the /TEMPVS/ that was shared by Subroutines BELTG and BELTRT. The COMMON BLOCK was commented out, all variables were explicitly typed and dimensioned. Only those variables that were shared by both subroutines [APA, UVA, DLGA, UAA, APB, UVB, DLGB, UBB] were place in the module. The remaining variables [TA, TB, TC, UP, B_BELT, UC, AX, XE, BX, ACA, ACB] were used only in Subroutine BELTG, and hence were made local variables of that subroutine.

MODULE_FLEXIBLE: – created from COMMON BLOCKs from the ATB code associated with the flexible segments. COMMON BLOCKs were commented out, all variables were explicitly typed and dimensioned.

MODULE_WATER: – created from COMMON BLOCKs from the ATB code associated with the water force option. COMMON BLOCKs were commented out, all variables were explicitly typed and dimensioned. Included /WMASS/, which was not included in BLOCK_DATA of Version 5.1. Added /TEMPFD/ that was shared by Subroutines ADDMAS and BOYCTR. The COMMON BLOCK was commented out, all variables were explicitly typed and dimensioned. All variables have been included at this point. Variables FF and UU were renamed to FF_WATER, and UU_WATER, respectively, to avoid any conflicts with the global variables FF and UU in Module STANDARD. Also renamed KSEG to KSEG_WATER to avoid conflicts with some local variables.

Selected Modified Version V.1 Subroutines and Functions

ADDMAS. The module MODULE_WATER_TEMP was created to contain the /TEMPFD/ common block, which is shared by Subroutine BOYCTR. Renamed KSEG to KSEG_WATER to avoid conflicts with similarly named variables in other subroutines

ADJUST – The local variable W was changed to WL to distinguish it from the W array in module MODULE_STANDARD.

AINPUT – Created from code taken from the main program related to reading in the A cards and opening files. The use of Hollerith types for TP1, TP8, UF1, and UF8 were replaced with character constants

AIRBAG - The temporary named COMMON /AIRBAGS_TEMPVS/ was created to handle the sharing of a /TEMPVS/ between Subroutines AIRBAG and AIRBGG.

AIRBG1 - The local variable MSEG was changed to MSEGL, to avoid a conflict with the array MSEG that exists in MODULE_STANDARD.

AIRBG3

AIRBGG.

APPLY - Renamed T2 to T2_LCL to avoid conflicts with MODULE_STANDARD

BELTG - The temporary named COMMON /BELTS_TEMPVS/ was created to handle the sharing of a /TEMPVS/ between Subroutines BELTG and BELTRT. Since [TA, TB, TC, UP, B_BELT, UC,AX, XE, BX, ACA, ACB] in /TEMPVS/ were used only in BELTG, they were removed from the module and made local variables of BELTG. The variable B was replaced with B_BELT to avoid a conflict with the B in MODULE_STANDARD. Also, the local variable GG was replaced with GG_BELT to avoid a conflict with GG in MODULE_STANDARD. Furthermore, the local variable BD was replaced with BD_BELTG to distinguish it from the variable BD in MODULE_STANDARD.

BELTRT - The temporary named COMMON /BELTS_TEMPVS/ was created to handle the sharing of a /TEMPVS/ between Subroutines BELTG and BELTRT, hence the use of /TEMPVS/ was deleted.

BGG - Renamed T2 to T2_LCL to avoid conflicts with MODULE_STANDARD. The use of /TEMPVS/ was eliminated, and the all the variables in /TEMPVS/, except the placing holding DUMMY array, were included as local variables. The following variables were rename to avoid confusion with the variables in MODULE_STANDARD: B to B_BGG, AB to AB_BGG, FF to FF_BGG, IFULL to IFULL_BGG, and VSCS to VSCS_BGG. The variable YFB was renamed to YFB_BGG to avoid a conflict with the STANDARD_FLEXIBLE module.

BINPUT - The variable array JOINTF was set = 0 directly instead of using two nested DO loops. The COMMON BLOCK /TEMPVS/ was replaced with local variables. Note that the

COMMON BLOCK /FXJROT/ was included with Version 5.1, but no variables are actually referenced from this common block in BINPUT.

BLKDTA - The COMMON BLOCK /TEMPVS/ was replaced with local variables. Added "D0" to integer constants, where needed.

BLOCK_DATA - replaced by MODULE_STANDARD, MODULE_FLEXIBLE, and MODULE_WATER.

BOYCTR -.The named COMMON /TEMPFD/, shared between Subroutines ADDMAS and BOYCTR, was included in MODULE_WATER. Note: COMMON /CONTRL/ was included in the Version 5.1 BOYCTR code, but no use was made of any of the variables in the common block. Renamed KSEG to KSEG_WATER to avoid conflicts with local variables in other subroutines. Also, renamed the local variable BB to BB_LCL to avoid a conflict with the global variable BB in /HRNESS/.

CFACTT

CHAIN - The common block /FXVAR/ was included in Subroutine CHAIN, but no variables were referenced. The common block /TEMPVS/ was eliminated, with all its variables redefined as local variables.

CHKRET - Common /BAGDIM/ had been moved to MODULE_STANDARD

CINPUT - Note: COMMON /ACTFR1/ was included in the Version 5.1 CINPUT code, but no use was made of any of the variables in the common block. The /TEMPS/ common block was placed in MODULE_STANDARD as global variables

CMPUTE

COMPSYS – Eliminated. Information placed in MODULE_STANDARD.

CONTCT - COMMON /ACTFR1/ was included in the Version 5.1 CONTCT code, but no use was made of any of the variables in the common block.

CONVERT_TIME - New subroutine that converts the elapsed time from the Fortran 90 intrinsic subroutine DATE_AND_TIME into hours, minutes etc.

CROSS

CUTARE - Renamed U1 to U1_LCL and V1 to V1_LCL to avoid conflicts with MODULE_STANDARD

DATE_TIME – New subroutine that gets the date and time of the run using the Fortran 90 intrinsic subroutine DATE_AND_TIME.

DAUX - Renamed T2 to T2_LCL to avoid conflicts with MODULE_STANDARD. Took the /TEMPVS/ common and placed it in MODULE_STANDARD. Placed /CMASS/ into MODULE_WATER.

DAUX11 - The variables DN, DM, SN, SM, HH, and BN were at the end of the original /TEMPVS/, the remaining variables are in /TEMPVS/ are in MODULE_STANDARD. They are apparently not needed in DAUX11, so they were deleted.

DAUX12 - The variables DN, DM, SN, SM, HH, and BN were at the end of the original /TEMPVS/, the remaining variables are in /TEMPVS/ are in MODULE_STANDARD. Only SN and SM

were need, they were made into local variables. The rest are apparently not needed in DAUX12, so they were deleted.

DAUX22 - The variables DN, DM, SN, SM, HH, and BN were at the end of the original /TEMPVS/, the remaining variables are in /TEMPVS/ are in MODULE_STANDARD. Only SN, BN, and HH were needed, which were made into local variables. The rest are apparently not needed in DAUX22, so they were deleted. Initialized HH directly (i.e. as an array =0, instead of using nested DO loops).

DAUX31 - The variables DN, DM, SN, SM, HH, and BN were not at the end of the original /TEMPVS/ in this subroutine

DAUX32 - The only the variables DN, DM, and BN were at the end of the original /TEMPVS/ in this subroutine, but they are not used in the subroutine, hence they were deleted.

DAUX33 - There were no extra variables at the end of the original /TEMPVS/ in this subroutine

DAUX44 - Note that the notes in the code state that /TEMPVS/ was not being shared by DAUX44, however, DAUX44 does use it. This use of /TEMPVS/ probably was added later, I think for the slip joints, but the note wasn't updated.

DAUX55 - Note that the notes in the code state that /TEMPVS/ was not being shared by DAUX55, however, DAUX55 does use it. This use of /TEMPVS/ probably was added later, I think for the slip joints, but the note wasn't updated.

DHHPIN - Initialized arrays BN and DD by direct assignment, i.e., eliminated the DO loops.

DINTG - Renamed the variable array F to FF to coincide with the renaming that occurred in /CDINT/ to avoid a conflict with F in /CMATRX/. Replaced XTEST with XTEST_SNGL and REGT with REGT_SNGL so that these are the single-dimensioned versions of XTEST and REGT. The arrays XTEST and XTEST_SNGL are equivalenced in MODULE_STANDARD, as are REGT and REGT_SNGL. Note, the comments in the code mentioned that XTEST is single dimensioned in Subroutine DINTG, but neglect to mention that REGT is also treated as single-dimensioned in Subroutine DINTG.

DOT31

DOT33

DOTT31

DOTT33

DOTVEC

DRCIJK - Variables D, ANG, and HT were renamed to D_LCL, ANG_LCL, and HT_LCL, respectively, to avoid any conflict with the global variables D, ANG, and HT.

DRCQUA - Variables D and E were renamed to D_LCL, and E_LCL, respectively, to avoid any conflict with the global variables D, and E.

DRCYPR - Variables D and B were renamed to D_LCL, and B_LCL, respectively, to avoid any conflict with the global variables D, and B.

DRGCHK - Renamed the variable H to H_LCL to avoid any conflicts with the H variable in MODULE_STANDARD. Renamed the variable QQ to QQ_LCL to avoid any potential

conflicts with the QQ variable in /CSTRNT/ in MODULE_STANDARD. Also renamed UU to UU_WATER as was done for /TEMPFD/ in MODULE_WATER_TEMP to avoid conflict with UU in MODULE_STANDARD.

DRIFT - Replaced /TEMPVS/ with local variables

DSETD - Changed D to D_LCL to avoid a conflict with D in MODULE_STANDARD.

DSETQ - Changed D to D_LCL and E to E_LCL to avoid a conflict with D and E in MODULE_STANDARD.

DSMSOL -. Changed B to B_LCL to avoid a conflict with B and in MODULE_STANDARD.

DZP - Changed E to E_LCL and GG to GG_LCL and W to W_LCL to avoid conflicts with E, GG, and W in MODULE_STANDARD.

ELTIME – Minor revisions.

EDEPTH - Changed AB to AB_LCL, B to B_LCL, C1 to C1_LCL and Y to Y_LCL to avoid conflicts with AB, B, C1, and Y in MODULE_STANDARD.

EFUNCT - Renamed JSTOP to JSTOP_LCL to avoid a conflict with JSTOP in MODULE_STANDARD.

EJOINT - Changed IJ to IJ_LCL to avoid conflicts with in MODULE_STANDARD. Changed the /TEMPVS/ common to local variables.

ELARE3 - Made the following variable name substitutions: FF to FF_WATER and RPH to RPH_LCL (for MODULE_WATER); NN to NN_LCL, PHI to PHI_LCL, and TQE to TQE_LCL (for MODULE_STANDARD). Also renamed local variables ASS to A1SS and ASS2 to A2SS for better code appearance. Set FRC and TQE_LCL = 0.0 without the use of a DO loop in one place, making use of the new array handling facility of Fortran 90.

ELAREA -. Made the following variable name substitutions: B to B_LCL, C to C_LCL, F to F_LCL, FF to FF_LCL, NN to NN_LCL, and PHI to PHI_LCL (for MODULE_STANDARD). FL to FL_LCL (for MODULE_WATER). Also renamed local variable ASS to A1SS for better code appearance. Changed all the /TEMPVS/ variables to local variables.

ELONG -. Made the following variable name substitutions: B to B_LCL, C to C_LCL, D to D_LCL, E to E_LCL, F to F_LCL, and G to G_LCL to avoid conflicts with the variables in MODULE_STANDARD.

EQUILB - Changed F1 to F1_LCL to avoid conflicts with in MODULE_STANDARD. Changed the /TEMPVS/ common to local variables.

ETA

EULRAD - Renamed B to B_LCL, D to D_LCL, and IC to IC_LCL to avoid conflicts with MODULE_STANDARD.

EVALFD - Renamed D to D_LCL and F to F_LCL to avoid conflicts with MODULE_STANDARD.

FDINIT - Renamed NTPL to NTPL_LCL and IFLAG to IFLAG_LCL to avoid conflicts with MODULE_STANDARD. The variables in /TEMPVS/ were placed in MODULE_STANDARD as global variables, with MS renamed to MS_SEG and NF renamed to NF_FUNCT.

FDUX11 - Changed T2 to T2_LCL to avoid conflicts with MODULE_STANDARD.

FDUX12

FDUX22

FILES

FINPUT - Changed IJK to IJK_LCL to avoid conflicts with in MODULE_STANDARD. Changed MS to MS_SEG, and NF to NF_FUNCT, which had been in the /TEMPVS/ common that was shared between Subroutines CINPUT, FDINIT, FINPUT, and HINPUT, and which was placed in MODULE_STANDARD.

FJNTVC - Changed T2 to T2_LCL and TMP1 to TMP1_LCL to avoid conflicts with in MODULE_STANDARD. Note: COMMON /FXCOEF/ was in the Version V.1 code, but no variables in it were referenced. Placed the common block /OLDDAT/ into the Module MODULE_FLEXIBLE.

FLXSEG - Changed CN to CN_LCL to avoid conflicts with MODULE_STANDARD. Note: all the variables in COMMON /TEMPVS/ were made into local variables.

FNAME – Changed F to F_IN, and E to E_IN to avoid conflicts with MODULE_STANDARD.

FNTERP - Changed PHI to PHI_LCL to avoid conflicts with MODULE_STANDARD

FRCDFL - Changed D to D_LCL and F to F_LCL to avoid conflicts with MODULE_STANDARD.

FSETUP - Changed T2 to T2_LCL to avoid conflicts with MODULE_STANDARD.

FSMSOL - Changed B, C, D, IJ, NN to B_LCL, C_LCL, D_LCL, IJ_LCL, NN_LCL to avoid conflicts with MODULE_STANDARD.

FSPDMP

FUNCHK

FXCAHW

FXCBLT - Changed NN to NN_LCL to avoid conflicts with MODULE_STANDARD.

FXCPLS

FXINPT - Used the PI from the MODULE_STANDARD instead of the one defined here for consistency in the code. In Version V.1, COMMON blocks /FXCOEF/ and /FXJROT/ were included, but no variables are used from these common blocks. Also, FORMAT # 100 was not referenced, so it was deleted.

FXMACC - Changed T2 to T2_LCL and T5 to T5_LCL to avoid conflicts with MODULE_STANDARD.

FXMASS - Changed T2 to T2_LCL to avoid conflicts with MODULE_STANDARD.

FXMODD - Changed NEQ to NEQ_LCL to avoid conflicts with MODULE_STANDARD.

FXMODV - Changed NEQ to NEQ_LCL and VAR to VAR_LCL to avoid conflicts with MODULE_STANDARD.

FXMODV0 - Changed NEQ to NEQ_LCL and VAR to VAR_LCL to avoid conflicts with MODULE_STANDARD.

GET_MONTH_NAME – New subroutine. Converts the number of the month obtained from a call to Fortran 90 intrinsic subroutine DATE_AND_TIME, to a name.

GLOBAL

HBELT - The /TEMPVS/ common was removed and placed in MODULE_STANDARD. The following variables were renamed to equate to the renamed variables from the /TEMPVS/ now in MODULE_STANDARD: T1 to T1_HRN, T2 to T2_HRN, R to R_HRN, V to V_HRN, S to S_HRN, E to E_HRN, U to U_HRN, TR to TR_HRN, ZR to ZR_HRN, FB to FB_HRN, FP to FP_HRN, BL to BL_HRN, FCE to FCE_HRN, FR to FR_HRN, T to T_HRN. Note: COMMON /FXVAR/ was included in Version 5.1, but no variables are actually used from it.

HBPLAY - The /TEMPVS/ common was removed and placed in MODULE_STANDARD. The following variables were renamed to equate to the renamed variables from the /TEMPVS/ now in MODULE_STANDARD: T1 to T1_HRN, T2 to T2_HRN, R to R_HRN, V to V_HRN, S to S_HRN, U to U_HRN, ZR to ZR_HRN, BL to BL_HRN, and NOLD to NOLD_HRN.

HEDING - The /TEMPVS/ common was removed and placed in MODULE_STANDARD. Note: COMMONs /ACTFR/ and /ACTFR1/ were included in Version 5.1, but no variables are actually used from them.

HERRON - Changed PHI to PHI_LCL and P1 to P1_LCL to avoid conflicts with MODULE_STANDARD. Also changed P2 to P2_LCL for consistency with P1.

HICCSI - The /CDINT/ common was deleted and the variables in it: JDTPTS, SPAN, IDBDY and Z were passed as arguments into the subroutine. Z was renamed to Z_LCL to free up the variable Z for global use. DT was renamed to DT_LCL, AREA was renamed to AREA_LCL, and HT was renamed to HT_LCL to avoid conflicts with global variables in MODULE_STANDARD. The variable EPS(6) was used to replace the explicit use of 1.0D-6 used in one of the IF tests.

HINPUT - The /TEMPVS/ common was removed and placed in MODULE_STANDARD. NF was renamed to NF_FUNCT to equate to the renamed variables from the /TEMPVS/ now in MODULE_STANDARD.

HPTURB -The /TEMPVS/ common was removed and placed in MODULE_STANDARD. The following variables were renamed to equate to the renamed variables from the /TEMPVS/ now in MODULE_STANDARD: T1 to T1_HRN, T2 to T2_HRN, R to R_HRN, E to E_HRN, T to T_HRN, RHS to RHS_HRN, IJK to IJK_HRN, C to C_HRN, and FCE to FCE_HRN. Removed equivalencing of BLOSS and HLOSS and placed in MODULE_STANDARD. Renamed SCALE to SCALE_LCL and IJ to IJ_LCL to avoid any conflicts with global variables.

HSETC - The /TEMPVS/ common was removed and placed in MODULE_STANDARD. The following variables were renamed to equate to the renamed variables from the /TEMPVS/ now in MODULE_STANDARD: TR to TR_HRN, FB to FB_HRN, U to U_HRN, V to V_HRN, B to B_HRN, S to S_HRN, BL to BL_HRN, FR to FR_HRN, R to R_HRN, E to E_HRN, T to T_HRN, RHS to RHS_HRN, IJK to IJK_HRN, C to C_HRN, and FP to FP_HRN. Renamed C1 to C1_LCL and IJ to IJ_LCL to avoid any conflicts with global variables.

HYABF – Changed F to F_LCL, and B to B_LCL to avoid conflicts with MODULE_STANDARD.

HYBND – Changed C to C_LCL, and U to U_LCL to avoid conflicts with MODULE_STANDARD.

HYBOX – Changed E to E_LCL, T2 to T2_LCL, D to D_LCL, CK to CK_LCL, P1 to P1_LCL, and F to F_LCL to avoid conflicts with MODULE_STANDARD.

HYDAD – Changed D to D_LCL to avoid conflicts with MODULE_STANDARD.

HYEST – All local variables were explicitly typed. Deleted /TEMPVS/ and used the /PLELP_TEMPVS/ variables for D12=>DMNT_PLP and renamed R to R_PLP. Made A, C, T, V and B into local variables. Renamed B to B_LCL and C to C_LCL. C_LCL, T and V are now passed as arguments to Subroutine HYLPX. The remaining variables that were in /TEMPVS/ are not used by the subroutine. Use the object-orientated nature of arrays to zero some of them. Changed BB to BB_LCL, F1 to F1_LCL, and TAB to TAB_LCL to avoid conflicts with MODULE_STANDARD.

HYFCN – Changed C to C_LCL to avoid conflicts with MODULE_STANDARD. Made a local parameter, Q_LIMIT, for an IF test.

HYLIM – Changed B to B_LCL, C to C_LCL, G to G_LCL, P1 to P1_LCL, P2 to P2_LCL, BD to BD_LCL, U to U_LCL, W to W_LCL, and E to E_LCL, to avoid conflicts with MODULE_STANDARD. Replaced a hard-wired 1x10-D10 with EPS(10) in an epsilon test for consistency. Made ITER into a local parameter for its IF test.

HYLPR – Changed C to C_LCL, and E to E_LCL, to avoid conflicts with MODULE_STANDARD.

HYLPX – The /TEMPVS/ common block was deleted. Only 3 variables within it are actually used and were renamed as follows: R to R_LCL and D to D_LCL. V was made into a local variable. All three variables (R_LCL, D_LCL, and V) are now passed into the subroutine as arguments. Changed B to B_LCL, W to W_LCL, E to E_LCL, and C to C_LCL to avoid conflicts with MODULE_STANDARD. Used new f95 option to initialize S array without using Do loops. Used the CYCLE statement.

HYNTR – Changed C1 to C1_LCL, CN to CN_LCL, BET to BET_LCL, F to F_LCL, P1 to P1_LCL, and TAB to TAB_LCL, to avoid conflicts with MODULE_STANDARD. Replaced the hardwired iteration count with the PARAMETE ITER_HYNTR. Replaced the hardwired 1x10-10 with EPS(10). Deleted /TEMPVS/ and replaced with the USE statement to /PLELP_TEMPVS/; using D12=>DMNT_PLP, and renamed R to R_PLP. Made AZ into a local variable, as well as A and B. Renamed B to B_LCL to avoid conflicts with MODULE_STANDARD. The other variables in /TEMPVS/ were not used.

HYPEN – Changed E to E_LCL, to avoid conflicts with MODULE_STANDARD.

HYREA – Changed BB to BB_LCL, H to H_LCL, AB to AB_LCL, and AREA to AREA_LCL to avoid conflicts with MODULE_STANDARD.

HYSOL

HYVAL – Replaced the hard-wired 0.000001's with EPS(6). Replaced the hard-wired 100 limit on iterations with MAXITER_HYVAL. Changed F1 to F1_LCL, U to U_LCL, BD to BD_LCL, and NSTEP to NSTEP_LCL, to avoid conflicts with MODULE_STANDARD.

HYVBX – Changed C to C_LCL, D to D_LCL, and B to B_LCL, to avoid conflicts with MODULE_STANDARD.

HYVFN – Changed C to C_LCL, U to U_LCL, F to F_LCL, and B to B_LCL, to avoid conflicts with MODULE_STANDARD.

IMPLS2 – The COMMON BLOCK, /CSTRNT/, was included in the subroutine in version V.1, but no variables were used from it. Changed H to H_LCL, and TN to TN_LCL, to avoid conflicts with MODULE_STANDARD. Deleted /TEMPVS/ and made all the variables local variables.

IMPULS

INITAL

INPROJ – Changed IJ to IJ_LCL to avoid any conflicts with MODULE_STANDARD.

INRTIA – Changed T1 to T1_LCL, T2 to T2_LCL, and T5 to T5_LCL, to avoid any conflicts with MODULE_STANDARD. Note: changed hardwired (32.114x12.0) to G of CNSNTS to maintain the flexibility of the code.

INTANG – Note that the COMMON BLOCK /VPOSTN/ was included in this subroutine in version V.1, but no variables were used from it.

INTERS – Renamed B to B_LCL and C to C_LCL to avoid conflicts with global variables in MODULE_STANDARD.

INTLIN – Renamed T2 to T2_LCL to avoid conflicts with global variables in MODULE_STANDARD.

INVERS – Renamed B to B_LCL and W to W_LCL to avoid conflicts with global variables in MODULE_STANDARD. Changed the local array, AA from AA(50,50) to AA(ND,ND) to ensure that the originally hardwired array, which stores array A, is always the same size as A.

JNTFNC – Renamed T2 to T2_LCL and T5 to T5_LCL to avoid conflicts with global variables in MODULE_STANDARD.

JNTROT – Renamed T2 to T2_LCL to avoid conflicts with global variables in MODULE_STANDARD. Placed the common block /OLDDAT/ into Module MODULE_FLEXIBILE.

KINPUT – Deleted the Common /TEMPVS/ and instead used the variables KTITLE and JTITLE from /CINPUT_TEMPVS/ in Module MODULE_STANDARD. Made the array TH, that had been in the /TEMPVS/ common block, into a local array. Replaced the old intrinsic function DFLOAT with DBLE. Renamed BLANK to BLANK_4 and changed it from a REAL type to a CHARACTER type.

L_LTIME – Renamed to avoid conflicts with intrinsic subroutines on some machines that are called LTIME. Calls the Fortran 95 intrinsic subroutine CPU_TIME.

LUDCMP – Replaced 2 cases of hard-wired epsilon tests with the EPS array. Renamed ORDER to I_ORDER so that its first letter is consistent with the default meanings.

LUPIVT – Renamed ORDER to I_ORDER so that its first letter is consistent with the default meanings. Also renamed SAVE to RSAVE to be different from the SAVE intrinsic function.

LUSUB – Renamed ORDER to I_ORDER and LU to R_LU so that their first letters are consistent with the default meanings. Also renamed B to B_LCL to avoid conflicts with variables in MODULE_STANDARD.

MAIN – Renamed the logical variables NPRT1, NPRT2, NPRT3 to L_NPRT1, L_NPR2, L_NPRT3, respectively, for clarity. Added a PROGRAM statement and made the END

statement a named END statement for clarity. Removed all the code related to input from the A cards and placed it in a new subroutine, AINPUT. Removed Hollerith usage in FORMAT 11 and in the call to Subroutine PRINT.

MAT31 – Deleted the PARAMETER statements, which serve no purpose anyway. Renamed B to B_LCL and C to C_LCL to avoid conflicts with variables in MODULE_STANDARD.

MAT33 – Renamed B to B_LCL and C to C_LCL to avoid conflicts with variables in MODULE_STANDARD. All DO loops that ended on a statement function had the termination number replaced with an END DO statement.

MUTPLY – Renamed B to B_LCL and C to C_LCL to avoid conflicts with variables in MODULE_STANDARD.

NUMCHR – Changed the CHARACTER typing from CHARACTER*(*) to CHARACTER (*).

ORTHO – Renamed D to D_LCL to avoid conflicts with variables in MODULE_STANDARD.

OUTPUT – Note that Common block /ACTF1/ was listed in the Version V.1 code but no variables were referenced from it. The /TEMPVS/ common was deleted and all the variables in it were made local variables. Apparently Subroutine HEDING, which is called by Subroutine OUTPUT, does not need any of the variables in the /TEMPVS/. Renamed IJK to IJK_LCL, USEC to USEC_LCL, T2 toT2_LCL and T5 to T_LCL to avoid conflicts with variables in MODULE_STANDARD. Found and corrected an apparent error. It appeared that if the number of airbags was not 0, then count of the number of tabular time histories would be incorrectly calculated because the count of the tabular time histories for the water forces would be skipped over. This occurred because there was a GO TO 960 in the section for the airbags that would skip over the water forces count for the tabular time histories if the number of airbags was not equal to 0. Furthermore, a computed GO TO statement was replaced with a CASE construct, which required duplicating some of the code instead of using GO TO statements, to avoid jumping into blocks. Made the array TDATA ALLOCATABLE to save a little bit of memory. The following still needs to be done: parameterize the use of '100.0' that is currently used to hardwire the scaling of several parameters.

PANEL – Renamed T2 to T2_LCL to avoid conflicts with MODULE_STANDARD.

PDAUX – Renamed NEQ to NEQ_LCL, MBAG to MBAG_LCL, DER to DER_LCL, and VAR to VAR_LCL to avoid conflicts with MODULE_STANDARD. Used REGT_SNGL instead of REGT because REGT was used in single dimension form. Deleted /TEMPVS/, made VXT a local variable. Note: T in /TEMPVS/ was not used in the subroutine.

PFDBOY – Note: the COMMON /SGMNTS/ was included in the V.1 code, but no variables are used from it. Renamed PHI to PHI_LCL and FF to FF_WATER to avoid conflicts with MODULE_STANDARD.

PFDFRC – Note: the COMMON /SGMNTS/ was included in the V.1 code, but no variables are used from it. Renamed PHI to PHI_LCL, TMP1 to TMP1_LCL, and FF to FF_WATER to avoid conflicts with MODULE_STANDARD. Note: TMP1 is computed but never used.

PFDWXC – Note: the COMMON /SGMNTS/ was included in the V.1 code, but no variables are used from it. Renamed PHI to PHI_LCL, TMP1 to TMP1_LCL, and FF to FF_WATER to avoid conflicts with MODULE_STANDARD. Note: TMP1 is computed but never used.

PLEDG – COMMON /TEMPVS/ was deleted and replaced with a USE statement, with some of its variables renamed with the suffix "_PLP" and placed in /PLELP_TEMPVS/, while other variables were made into local variables (refer to the section on /PLELP_TEMPVS for

details]. The renamed local variables from /TEMPVS/ are: AB to AB_LCL, AREA to AREA_LCL, B to B_LCL, BB to BB_LCL, E to E_LCL, and U to U_LCL. Renamed the local variables BD to BD_LCL and PL to PL_LCL to avoid conflicts with MODULE_STANDARD.

PLELP – COMMON /TEMPVS/ was deleted with some of its variables renamed with the suffix "_PLP" and placed in /PLELP_TEMPVS/, while other variables were made into local variables (refer to the section on /PLELP_TEMPVS for details]. The renamed local variables from /TEMPVS/ are: UH_LCL, TD_LCL, and XH_LCL. Renamed the local variables NN to NN_LCL and BET to BET_LCL to avoid conflicts with MODULE_STANDARD.

PLREA – Renamed AB to AB_LCL, AREA to AREA_LCL, BB to BB_LCL, C to C_LCL, D to D_LCL, E to E_LCL, and H to H_LCL to avoid conflicts with MODULE_STANDARD.

PLSEGF – Common /FXVAR/ was listed in the V.1 code, but no variables were used from it. COMMON /TEMPVS/ was deleted with some of its variables renamed with the suffix "_PLP" and placed in /PLELP_TEMPVS/, while other variables were made into local variables (refer to the section on /PLELP_TEMPVS for details). The renamed local variable from /TEMPVS/ is: FR_LCL. The other variables from /TEMPVS/ made into local variables are: CF, VRM, VRT, VRTS, VRTEST, ELOSS, WCM, WCN, TQM, TQN, TQNT, and VMN. Renamed the local variable FF to FF_LCL to avoid conflicts with MODULE_STANDARD. Note: should make VRTEST into a parameter.

POSTPR - The /TEMPVS/ was replaced with a USE statement for /HEDING_TEMPVS/. Used array object form to initialize several variables, e.g. A = 0, where A(I,J). The /CDINT/ common was deleted and the variables in it: JDTPTS, SPAN, IDBDY and ZZ were made into local variables and passed to Subroutine HICCSI as new arguments. Array ZZ was made ALLOCATABLE. Added a somewhat ineffectual test for NHIC, as well as an associated STOP 478. Fixed logic so of the loop so that Subroutine E_ELTIME is properly initialized.

PRINT – Renamed T2 to T2_LCL, and MBAG to MBAG_LCL to avoid conflicts with MODULE_STANDARD. Note: Common block /ACTFR1/ was listed in the V.1 code, but no variables were used from it. Deleted /TEMPVS/, and made all the variables that were in the common block local variables.

PRNCIPAL - Note that this subroutine had IMPLICT REAL*8 (A-Z), which is inconsistent with all the over 100 other subroutines which had IMPLICT REAL*8 (A-H,0-Z). Therefore, to keep some consistency all the variables beginning with the letters I-N, were renamed with a preface of "R_". These variables are: R_I, R_L, R_M, R_N, R_I1, R_I2, R_I3, R_R_IXX, R_IYY, R_IZZ, R_IXY, R_IYZ, and R_IXZ. In addition, all the integer values, such as 1, 2 etc., were made into explicit real constants, e.g. 1.0 for the sake of clarity. Also, comments describing basic operations, such as how to normalize a vector, were deleted. Furthermore, the local definition of PI was deleted and replaced with replaced with a USE statement referencing the global definition of PI used by the rest of the program. Deleted many unnecessary parentheses. Added a missing RETURN statement. Changed the one WRITE statement from logical unit 5 to logical unit 6, as well as putting the outputted statement into a format statement and gave it carriage control.

QSET – Renamed F to F_LCL, T2 to T2_LCL, DER to DER_LCL and Y to Y_LCL to avoid conflicts with MODULE_STANDARD.

QUAT – Renamed R to R_LCL and ANG to ANG_LCL to avoid conflicts with MODULE_STANDARD. Renamed DOT to DOT_LCL to avoid confusion with subroutine of the same name.

RCRT – Renamed W to W_LCL, PL to PL_LCL and S1 to S1_LCL to avoid conflicts with MODULE_STANDARD. Used the matrix nature of T to initialize it.

ROBINP

ROT – Renamed C to C_LCL to avoid conflicts with MODULE_STANDARD.

ROTATE – COMMON /TEMPVS/ was deleted and all its variables were made into local variables. The common /ACTFR1/ was listed in the V.1 version but no variables from it were referenced. Renamed IJ to IJ_LCL, T2 to T2_LCL and T5 to T5_LCL to avoid conflicts with MODULE_STANDARD. Changed the code for the water forces from lower case to upper case. Added the STOP number 449 to the wind force section of the code because the STOP was not numbered.

SEGSEG – COMMON /TEMPVS/ was deleted with some of its variables renamed with the suffix "_PLP" and placed in /PLELP_TEMPVS/, while other variables were made into local variables (refer to the section on /PLELP_TEMPVS for details]. The renamed local variables from /TEMPVS/ are: B_LCL, T1_LCL, T2_LCL, T3_LCL, T4_LCL, TEMP_LCL, TT4_LCL, and TT5_LCL. Renamed the local variables BET to BET_LCL, NN to NN_LCL, NS to NS_LCL, and S1 to S1_LCL to avoid conflicts with MODULE_STANDARD.

SETUP1 – The following COMMON /TEMPVS/ variables were made into local variables; S, S1, S2, SR2, T, T1, T2, T3, T4, T5, T6, VIT, while the following variables were deleted because they were not used: T7, T8, T9, T10, T11, T12, HH, TT1, TT2, SQS1, S3, S4, SR2. Renamed T2 to T2_LCL and T5 to T5_LCL to avoid conflicts with MODULE_STANDARD.

SETUP2 – The all the COMMON /TEMPVS/ variables were made into local variables except S and T12, which were not used in the subroutine. Renamed FORCE to FORCE_LCL, S1 to S1_LCL, T2 to T2_LCL and T5 to T5_LCL to avoid conflicts with MODULE_STANDARD.

SIMPSN – Renamed F to F_LCL and B to B_LCL to avoid conflicts with MODULE_STANDARD.

SINPUT – Common block /ACTFR1/ was included in the V.1 code but was not referenced. All the COMMON /TEMPVS/ variables were made into local variables. Renamed S1 to S1_LCL, and P1 to P1_LCL to avoid conflicts with MODULE_STANDARD.

SOLVA – Renamed A11 to A11_LCL, A13 to A13_LCL, A22 to A22_LCL, and A23 to A23_LCL to avoid conflicts with MODULE_STANDARD.

SOLVR – Renamed B to B_LCL, D to D_LCL, and T2 to T2_LCL, to avoid conflicts with MODULE_STANDARD.

SPDAMP – Note: Common /FXVAR/ was listed in the V.1 code, but was not referenced. All COMMON /TEMPVS/ variables were made into local variables. Renamed T2 to T2_LCL and T5 to T5_LCL to avoid conflicts with MODULE_STANDARD.

SPLINE – Renamed C to C_LCL, F to F_LCL and Y to Y_LCL to avoid conflicts with MODULE_STANDARD.

SPRNGF - Renamed D to D_LCL, JSTOP to JSTOP_LCL, U to U_LCL, and Y to Y_LCL to avoid conflicts with MODULE_STANDARD.

TILDE

TRIGFS

TRNPOS

U1ASC – Added a SAVE statement, which is redundant (since the DATA statement implies this attribute) but adds to the clarity of the code.

U1OLD –Replaced the TEMPVS common block with all local variables. Deleted some unnecessary IF tests on NWATER. Added SAVE statement for clarity.

U1ASCD – Put commons /COUTN/, /COUTFMT/, and /BAGDIM/ into MODULE_STANDARD. Replaced the TEMPVS common block with all local variables, most of the variables that were in the TEMPVS common block were not needed and hence deleted. Deleted some unnecessary IF tests on NWATER.

U1ASCH – Subroutine was a mess, with many inconsistencies and gross mistakes. Note: the following commons were included in V.1, but not used: /CSTRNT/, /FORTOR/, /RSAVE/, SGMNTS. Replaced /TEMPVS/ with local variables except for the following variables that were in /TEMPVS/ but were not needed and hence deleted: XD, XSEGLP, T1, T2, T3, T4, T5, XBAR, XT3, XRK1, XRK2, XVDAT, XDUMMY, IMPL. The date of the run that was in /TITLES/, was changed back to DATE from DATEN to coincide with the definition in MODULE_STANDARD. Defined all the "X..." arrays that were in /TEMPVS/ as REAL, instead of the incorrect default of double precision that was in V.1.

UNIT1 – Changed double quotes to single quotes.

UNTVEC

UPDATE – Note: common blocks /ACTFR/ and /FXJROT/ were included in the Version V.1 code but no variables were referenced from them.

UPDFDC

UPDPFD – Renamed TQE to TQE_LCL to avoid conflicts with global variables in MODULE_STANDARD.

USER – Note: Common ACTFR was listed in V.1, but no reference was made to it.

VECANG

VECMAG

VEHPOS – All the variables in the /TEMPVS/ were made into local variables.

VINITL – Note that common /CONTRL/ was referenced and apparently NGRND was used from it in previous versions, but is currently not used. Changed the dimension IVREF from 6 to MAXVEH for consistency. All the variables in the /TEMPVS/ were made into local variables. NUMVEH was renamed to NUMVEH_LCL, and IVREF was renamed to IVREF_LCL to avoid conflicts with MODULE_STANDARD.

VINO12 – All the variables in the /TEMPVS/, as was well as the related parameters, were placed in MODULE_STANDARD. NUMVEH was renamed to NUMVEH_LCL, and F1 was renamed to F1_LCL to avoid conflicts with MODULE_STANDARD. The parameter CNV_MSEC was created to make the conversion of time from seconds to milliseconds clearer.

VINO34 – Most of the variables in the /TEMPVS/, was well as the related parameters, were replaced by their definition in /VIN_TEMPVS/ in MODULE_STANDARD. However, T3L, T3A, and T33 were in the V.1 code but not referred to. The parameter VT4 was listed but were not used. These variables were eliminated. Several other variables that were at the end of the /TEMPVS/, were made into local variables. NUMVEH was renamed to NUMVEH_LCL, and

MSEG was renamed to MSEG_LCL to avoid conflicts with MODULE_STANDARD. The parameter CNV_MSEC was created to make the conversion of time to msec clearer.

VINPUT – Most of the variables in the /TEMPVS/, was well as the related parameters, were replaced by their definition in /VIN_TEMPVS/ in MODULE_STANDARD.

VINTST – Most of the variables in the /TEMPVS/, was well as the related parameters, were replaced by their definition in /VIN_TEMPVS/ in MODULE_STANDARD. NUMVEH was renamed to NUMVEH_LCL to avoid conflicts with MODULE_STANDARD.

VISCOS – Renamed HA to HA_LCL to avoid conflicts with MODULE_STANDARD.

VISPR – The /TEMPVS/ was deleted and all the variables were made into local variables, except T8, which was not used in the subroutine. Note that /FXVAR/ was included in V.1, but no variables were referenced from it. Renamed IJ to IJ_LCL to avoid conflicts with MODULE_STANDARD. Treated T3, T6, T9, and ANGL arrays as objects to initialize them, i.e. T3 = 0.0 instead of using a DO loop. Replaced "1H0" with "'0'" in 2 FORMAT statements.

VPATH – Renamed NGRND to NGRND_LCL, NPG to NPG_LCL, NVEH to NVEH_LCL, NUMVEH to NUMVEH_LCL and SEG to SEG_LCL to avoid conflicts with MODULE_STANDARD.

VPATH2 – NUMVEH was renamed to NUMVEH_LCL, IVREF was renamed to IVREF_LCL, and NGRND was renamed to NGRND_LCL to avoid conflicts with MODULE_STANDARD.

VSPLIN – Variables in /TEMPVS/ that had been placed in /VIN_TEMPVS/ in MODULE_STANDARD, were used, with remaining variables that had been in /TEMPVS/, that were needed, were made into local variables. The variable F was renamed to F_LCL to avoid conflicts with MODULE_STANDARD. Made the arrays: F_LCL, Q1, SP, TT, and XYZ allocatable, allocated them and then deallocated them on exit from the subroutine to save space. The parameter MAXVT3 was listed in the comments in V5.1 as being needed but was not actually used. Furthermore, the variable THET was listed in the TEMPVS but was not actually used in the subroutine and hence was eliminated.

WATHED – Note that common blocks /CONTRL/ and /TITLES/, were listed in V.1, but no variables from it were referenced. Eliminated all the variables used by /TEMPVS/, except TDATA and USEC, which are now coming into the subroutine by the use of /HEDING_TEMPVS/. Renumbered all the FORMAT statements.

WATINP – Renamed KSEG to KSEG_WATER because it had been renamed for /TEMPFD/. Merged the format statements into the body of the code. Eliminated format statements 1600 and 3900 because no reference was made to them. Eliminated the original 100 CONTINUE statement because no reference was made to it in the subroutine. Renumbered all the FORMAT statements for clarity. Gave the STOP statement a character expression, even though it needs a number. Also renamed TMP1 to TMP1_LCL to avoid conflicts with MODULE_STANDARD.

WATOUT – Note that common block /CONTRL/, was listed in V.1, but no variables from it were referenced. Eliminated all the variables used by /TEMPVS/, except TDATA, which is now coming into the subroutine by the use of /HEDING_TEMPVS/. Renamed F to F_LCL, FF to FF_LCL, KELL to KELL_LCL, TMP to TMP_LCL, USEC to USEC_LCL, and WF to WF_LCL to avoid conflicts with other similarly name global variables. Also, renamed TMP1 to TMP1_LCL to avoid a conflict with global variables. Treated arrays AF, BF, DF, WEF, F_LCL, FF_LCL as objects in many places, i.e. eliminated some DO loops to take advantage of the object nature of Fortran 90. Format statement 1000 was eliminated because it was not used. Renumbered all the FORMAT statement and DO loops that remain.

WATSET

WAVEL - Renamed TMP1 to TMP1_LCL to avoid a conflict with global variables.

WBAREA – Note that common blocks /CONTRL/, /CNSNTS/, /CNTSRF/ and /WATINF1/ were listed in V.1, but no variables from them were referenced. Renamed KEG to KEG_WATER and UU to UU_WATER to avoid conflicting with a similarly named global variable. Also, renamed TMP1 to TMP1_LCL and V3 to V3_LCL to avoid a conflict with global variables. Note: hard-wired variables should be replaced with an EPS test

WELFOR – Note that common blocks /CONTRL/, and /WATINF2/ were listed in V.1, but no variables from them were referenced. Renamed KEG to KEG_WATER to avoid conflicting with a similarly named global variable. Also, renamed TMP1 to TMP1_LCL and TQE to TQE_LCL to avoid a conflict with global variables. Note: a hard-wired variable should be replaced with an EPS test.

WEXPHI – Note that common block /CNSNTS/ was listed in V.1, but no variables from it were referenced.

WFORCE – Note that common blocks /CONTRL/, /RSAVE/, and /WATGRD/ were listed in V.1, but no variables from them were referenced. Renamed UU to UU_WATER and KEG to KEG_WATER to avoid conflicts with other similarly name global variables. Also, renamed TMP1 to TMP1_LCL to avoid a conflict with global variables.

WINDY – Common /TEMPVS/ was deleted and all variables made into local variables. Renamed AREA to AREA_LCL, BET to BET_LCL, BTE to BTE_LCL, C to C_LCL, FF to FF_LCL, NN to NN_LCL, NSTEPS to NSTEPS_LCL, SCALE to SCALE_LCL, and Y to Y_LCL to avoid conflicts with other similarly name global variables. The DO 21 block and several others were eliminated and replaced with object equalities instead of by components. Moved the definition of MOELP closer to where it is used. Deleted the section on the forces and torques and replaced with Subroutine FORCE_TORQUE. Defined the parameter FT_SCALE to make it clearer where it is used. Used the VECMAG function in place of explicitly doing it in the subroutine.

XDY – Variable D was renamed to D_LCL, to avoid conflicts with MODULE_STANDARD.

YPRDEG – Variable D was renamed to D_LCL, to avoid conflicts with MODULE_STANDARD.